Programming Contest results p. 78

DISK ENCLOSED

AC's TECH AMIGA

For The Commodore

Volume 4 Number 1
US $14.95 Canada $19.95

Artificial Life

- Random Number Generation
- Artificial Life Programs
- Complex Functions in Assembly
- Function Genie's for ProDraw

Source & Executables ON DISK!

01
0 74470 78640 5

Special Offer for AC Readers!

AMOS (US), AMOS Compiler, and AMOS 3D

all three for only $99.99*

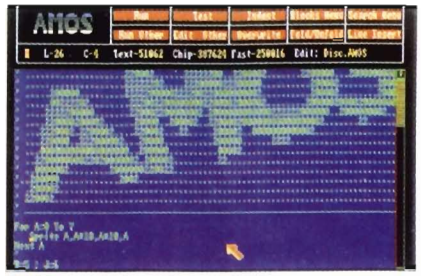
Bring your Amiga to *Life!*

AMOS - The Creator is like nothing you've ever seen before on the Amiga. If you want to harness the hidden power of your Amiga, then AMOS is for you!
AMOS Basic is a sophisticated development language with more than 500 different commands to produce the results you want with the minimum of effort. This special version of AMOS has been created to perfectly meet the needs of American Amiga owners. It includes clearer and brighter graphics than ever before, and a specially adapted screen size (NTSC).

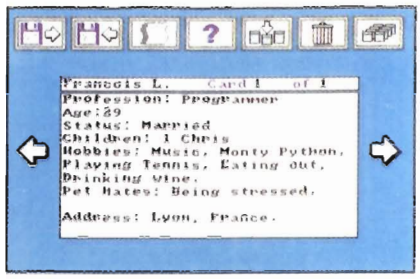
"Whether you are a budding Amiga programmer who wants to create fancy graphics without weeks of typing, or a seasoned veteran who wants to build a graphic user interface with the minimum of fuss and link with C routines, AMOS is ideal for you." *Amazing Computing, June 1992*

HERE ARE JUST SOME OF THE ▼ THINGS YOU CAN DO ▼

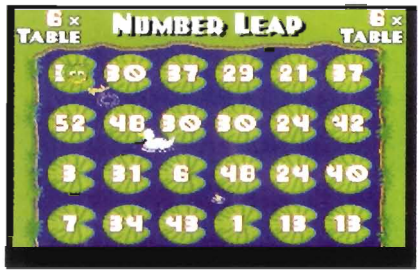
- Define and animate hardware and software sprites (bobs) with lightning speed.
- Display up to eight screens on your TV at once - each with its own color palette and resolution (including HAM, interlace, half-brite and dual playfield modes).
- Scroll a screen with ease. Create multi-level parallax scrolling by overlapping different screens - perfect for scrolling shoot-em-ups.
- Use the unique AMOS Animation Language to create complex animation sequences for sprites, bobs or screens which work on interrupt.
- Play Soundtracker, Sonix or GMC (Games Music Creator) tunes or IFF samples on interrupt to bring your programs vividly to life.
- Use commands like RAINBOW and COPPER MOVE to create fabulous color bars like the very best demos.
- Transfer STOS programs to your Amiga and quickly get them working like the original.
- Use AMOS on any Amiga from an A500 with a single drive to the very latest model with hard disk.



Use the sophisticated editor to design your creations



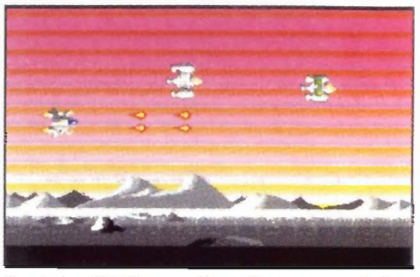
Create serious software like Dataflex



Produce educational programs with ease


Play Magic Forest and see just what AMOS can do!

Design sprites using the powerful Sprite Editor


Create breathtaking graphical effects as never before

WHAT YOU GET!

AMOS (US)—AMOS BASIC, sprite editor, Magic Forest and Amosteroids arcade games, Castle AMOS graphical adventure, Number Leap educational game, 400-page manual with more than 80 example programs on disk, sample tunes, sprite files, and registration card.

AMOS Compiler—AMOS Compiler, AMOS language updater, AMOS Assembler, eight demonstration programs which show off the power of the compiler, and a comprehensive, easy-to-use manual to develop lightning fast software.

AMOS 3D—Object Modeler, 30 new AMOS commands, and more. AMOS 3D allows you to create 3D animations as fast as 16 to 25 frames per second. You can display up to 20 objects at once, mix 3D with other AMOS features such as sprites, bobs, plus backgrounds, and more.

Limited Time Offer for AC readers only!

Get all three AMOS packages at one great price. Order today by sending your name, address (physical address please—all orders will be shipped by UPS), and $99.99 (*plus $10.00 for Shipping and handling) to: AMOS Special, PiM Publications, Inc., P.O. Box 2140, Fall River, MA 02722-2140 or use your VISA, MasterCard, or Discover and fax 1-508-675-6002 or call toll free in the US or Canada:

1-800-345-3360

Please allow 4 to 6 weeks for delivery.
AMOS written by François Lionet.

Country of Origin: UK

EUROPRESS SOFTWARE

AC's TECH/AMIGA

ADMINISTRATION

Publisher:	Joyce Hicks
Assistant Publisher:	Robert J. Hicks
Administrative Asst.:	Donna Viveiros
Circulation Manager:	Doris Gamble
Asst. Circulation:	Traci Desmarais
Traffic Manager:	Robert Gamble
Marketing Manager:	Ernest P. Viveiros Sr.

EDITORIAL

Managing Editor:	Don Hicks
Editor:	Jeffrey Gamble.
Hardware Editor:	Ernest P. Viveiros Sr.
Senior Copy Editor:	Paul Larrivee
Copy Editor:	Elizabeth Harris
Video Consultant:	Frank McMahon
Illustrator:	Brian Fox

ADVERTISING SALES

Advertising Manager: Wayne Arruda

1-508-678-4200
1-800-345-3360
FAX 1-508-675-6002

AC's TECH For The Commodore Amiga™ (ISSN 1053-7929) is published quarterly by PIM Publications, Inc., One Currant Road, P.O. Box 2140, Fall River, MA 02722-2140.

Subscriptions in the U.S., 4 issues for $44.95; in Canada & Mexico surface, $52.95; foreign surface for $56.95.

Application to mail at Second-Class postage rates pending at Fall River, MA 02722.

POSTMASTER: Send address changes to PIM Publications Inc., P.O. Box 2140, Fall River, MA 02722-2140. Printed in the U.S.A.

First Class or Air Mail rates available upon request. PIM Publications, Inc. maintains the right to refuse any advertising.

PIM Publications Inc. is not obligated to return unsolicited materials. All requested returns must be received with a Self Addressed Stamped Mailer.

Send article submissions in both manuscript and disk format with your name, address, telephone, and Social Security Number on each to the Editor. Requests for Author's Guides should be directed to the address listed above.

AMIGA™ is a registered trademark of Commodore-Amiga, Inc.

Contents

Volume 4 Number 1

3 **Artificial Life**
by John Iovine

8 **Huge Numbers**
by Michael Greibling

20 **PasteUp**
by J.T. Steichen

29 **GaugeClass**
by Scott Palmateer

42 **Programming the Amiga in Assembly Language: Complex Functions**
by William P. Nee

53 **Pseudo-Random Number Generation**
by Christopher Jennings

58 **Draw 5.0**
by T. Darrel Westbrook

70 **Writing a Function Genie for ProDraw**
by Keith D. Brown

78 **AC's TECH Programming Contest Results**

Departments

2 **Editorial**
40 **List of Advertisers**
41 **Source and Executables ON DISK!**

Startup Sequence

The Amiga is so much more.

As an *AC's TECH* reader, we believe you are interested in programming and building things for your Amiga. This means you want all the programming and hardware articles possible. So do we.

As one of the only sources for this base-level type of information, *AC's TECH* is constantly on the search for interesting articles. Recently we have seen an explosion in new submissions for *AC's TECH*. But, even with all this new material, we always need more.

Since our business is to provide both advanced and beginning Amiga developers with a disk-based magazine capable of displaying all facets of the Amiga's potential, it is essential that this information be as diverse as the Amiga itself. This is no small task. There are hundreds of areas where the Amiga can excel and very few of these areas have yet to be tapped.

More than video

The Amiga has attained enormous success as a video tool, however, the Amiga is much more than a video computer. Please do not misunderstand. Every Amiga user is very grateful for the acceptance the Amiga has achieved in the video industry. To many videophiles, there is no greater computer than the Amiga. Both Apple and IBM have watched the Amiga's markets closely and are now attempting to penetrate these Amiga strongholds. Yet the same qualities that make the Amiga a superb video tool can now be used in other areas.

The Amiga's multitasking environment and graphics capability, as well as its compatibility with existing television technology, not only spawned such hot selling ideas as NewTek's Video Toaster, but offered Scala an opportunity to introduce a full-range program called InfoChannel.

With InfoChannel, a network of Amigas can be established in a variety of locations to present specialized demonstrations. New programs or additional bits of programming information can be downloaded to the individual sites overnight by modem. Important updates and additional daily facts can be supplied to the system while it is performing. One master program can be established and updated while small changes can be applied at the local level. The satellite divisions can also upload their information to be tailored into the overall distribution. With InfoChannel, a company can continually update its work force on problems, solutions, opportunities, etc.

Scala has been able to install similar systems with different operating parameters into hotels, hospitals, corporations and more. But even as interesting as InfoChannel is, there are far more things the Amiga can do.

Amiga opportunities

When we discuss the possibilities of graphics-based computers in today's highly video-oriented industry, we tend to narrow our thinking to video or games. Yet the same high-quality graphics, sound, and multitasking environment needed to enhance these products will also do wonders for software in education, business, radio, industrial applications, engineering, design, manufacturing, publishing, and more.

Even as competing computers rapidly decrease prices, the Amiga is still able to win. No other platform has yet been able to put together the overall performance of an Amiga. It is this advantage that makes CD^{32} a very strong contender.

With more emphasis on set-top appliances for use in the many proposals for the "digital highway" the clear choice is CD^{32}. Not only does it offer a video-based technology and multitasking environment, but it is based on eight years of established development. This eight-year period has allowed a device such as CD^{32} to be introduced with a confidence in its ability to function error free. Eight years has also allowed the Amiga to develop many powerful programming tools.

The Amiga infrastructure

No Amiga product was ever created in a vacuum. In the beginning of the Amiga market (those early days at the end of 1985 and the beginning of 1986), only two types of products seemed available. In software, it was communications or terminal programs, and in hardware, it was memory expansion devices. Many developers saw these as viable products because the available development tools made them quick to create and these products would be needed by the growing Amiga marketplace.

As the Amiga matured, more products were introduced. Each new introduction created a new direction for the Amiga as well as a broadening array of development tools. Through time, both the development languages and their assorted tools continued to grow. The end result is an arsenal of development "tricks" for the Amiga.

It is this rich infrastructure that makes the Amiga such a valuable commodity today. With these tools and techniques, programmers and hardware designers are able to cut hundreds of man-hours out of their development schedule. The value of any computer platform rests squarely with the strength of its development infrastructure.

An offer

AC's TECH has a vested interest in seeing these tools developed further. We want to make these tools available to the general public through the pages of this publication. We want to know how you are using the Amiga, and inspire all of our readers to attempt even wider use of the Amiga.

We have just ended a programming contest (congratulations to the winners listed in this issue) with a notable amount of submissions. What we would like to do now is inspire every programmer and hardware hacker in the Amiga community to submit their creations.

If you have created a product you feel you want to market, write us and tell us what you have done. We will do our best to publicize your achievement. Remember, free advertising is worth twice what you pay for it.

At the same time, write us and tell us how you use your Amiga, what tools you have developed, or what programs you have initiated. We will reprint as many stories in *AC's TECH* and *Amazing Computing* as possible.

Our goal is two-fold. First, we want to place as many tools in the hands of aspiring developers as possible. Second, we want to inspire our readers by presenting the accomplishments of their fellow Amiga users.

While the pay is never large, we will offset that by periodically awarding prizes to the best submissions. The prizes will be those submitted by Amiga companies who also want to inspire you.

A few rules

Remember that this offer is void where prohibited or restricted by law, all submissions will be reviewed by judges whose decisions will be final, and all taxes will become the responsibility of the winners of any prizes awarded.

If a program or utility is submitted, please include the source code and (if warranted) the executable file.

Please submit all creations with a brief description of the project, its uses, your name and telephone number and your social security number (so we can pay you if your piece is published as a feature).

If you are submitting a brief description of your work with the Amiga, please include an address and telephone number with a time of day when we can contact you.

The Amiga is only as good as its foundation. The Amiga community has worked for years to provide the Amiga with a very good foundation. With your assistance, we can inspire each other to do even more.

Sincerely,

Don Hicks
Managing Editor

Artificial Life

by John Iovine

Creating artificial life forms is one of the most interesting things you can do with your Amiga computer. Artificial life programs are also called Cellular Automation (CA) programs and Genetic algorithms. Whatever you call them, this form of life is electronic and exists only within the confines of the computer system.

Artificial life programs are not merely toys for the technically inclined. These programs are used to successfully model biological organisms and eco-systems. Cellular automations can mimic evolution, co-evolution, the migration of birds, colonies of ants, social order of bees, and bacteria colonies. Chaos algorithms may also be incorporated into the CA programs to add a degree of randomness that increases the accuracy in modeling weather patterns, population growth, and the spread of infectious disease. Artificial life programs are also being developed to optimize neural networks, artificial intelligence, and parallel processors. Computer experiments are underway generating CA programs that will create and wire neural networks patterns. When these programs become successful, they essentially will teach themselves.

Genetic Algorithms

Genetic Algorithms are a class of artificial life that evolves in a Darwinian, survival-of-the-fittest, manner. I specify *Darwinian* because other researchers are busy using CA programs to test co-evolution theories. Typically Genetic Algorithms programs are designed to internally optimize their program code for specific problem solving.

Danny Hills, the founder of Thinking Machines Corporation in Cambridge, Massachusetts, programmed an interesting GA. In Hills Genetic Algorithm experiment he needed one of his company's Connection Machine computers. This super computer contains 65,536 parallel processors that allowed it to quickly simulate 65,536 independent organisms. The artificial organisms used in the experiment were numerical sorting programs.

To mimic evolution the programs could randomly change (mutate) their code. They could also combine their program codes with other program codes (organisms), in a process similar to artificial sex. All new programs created were tested for efficiency in sorting numbers. Only the fittest or most efficient programs survived. This electronic evolution scheme quickly evolved improvements in program code.

Writing Music

Artificial life programs can also write music. Eric Iverson of New Mexico State University created an artificial life program that creates music. This program is not like other music generation programs. Most computerized music generation programs rely on generating a large amount of random notes that are then filtered though an artificially intelligent (AI) sub-routine. The AI sub-routine section contains a bunch of "rules" that constitute the programmers best ideas of what note sequence can make music.

Artificial life music generation operates differently. The artificial life program eats music fed to it. We could feed the program any type of music from Madonna to Mozart. The program uses the music structure (notes) to generate new structures of notes.

The process I used in the BASIC program could easily be modified to generate new music from old music, although the BASIC algorithm is different from the one used by Mr. Iverson.

Cooperative Co-Evolution

Darwin's theory of evolution specifies survival of the fittest, not the nicest. Two Austrian mathematicians decided to have another look at this premise and have discovered, through the use of CA programs, that cooperative and unselfish behavior may play a role in the story of evolution. Karl Sigmund and Martin Nowak's new cooperative co-evolution theory is supported in nature.

For instance, vampire bats will share their blood meals with unrelated, less fortunate neighbors. A vampire bat is required to consume 50 to 100 percent of its body weight in blood every night. If it fails to feed for two nights in a row, it will die. However, a bat can

gain another 12 hours of life and anoth
regurgitated blood meal by a roost ma
food sharing, their annual mortality w
sharing the number drops down to 24
evolutionary standpoint this unselfish
propagation of the species. Bats are no
unselfish behavior.

Cellular automation programs va
Obviously Danny Hill's Connection M
complex for us to program onto our si
However, CA programs generally con
When the computer rapidly and consis
through generations, complex patterns
written into the program; they develop
striking behavioral patterns provide w
study and to model living systems and

To help you acquire a knowledge
programs, artificial life and genetic alg
BASIC language program listings to ge
language programs, although slow and
into these programming beasties. Thes
remain hidden in more complex, comp

Enter Life

No discussion of artificial life would be complete without mentioning John von Neumann's seminal work in the late 1940s on cellular automata. Martin Gardner's "Mathematical Games" column in *Scientific American* brought cellular automation to the masses. The "Game of Life" was originally described in *Scientific American* in the early 1970s by Martin Gardner. John Horton Conway, a British mathematician then at the University of Cambridge in England, created the game. Professor Conway is currently teaching at Princeton University.

The game is played on a large two-dimensional grid of square cells, similar to a checkerboard. Each cell or block of the checkerboard can contain an organism, identified as a dot. Each cell on the checkerboard is surrounded by eight neighboring cells. Four cells are adjacent orthogonally and four cells adjacent diagonally.

In the game of life, an organism dies or reproduces according to a few simple rules derived by Conway. In deriving rules, Conway sought to find the simplest rules that would create populations with the largest diversity and unpredictability. The rules to play one game of life are as follows:

1. Survival. Every organism with two or three neighboring cells that also contain organisms survives to the next generation.
2. Death. Each organism with four or more neighboring organisms dies of overpopulation. Every organism with just one neighbor or no neighbor dies of isolation.
3. Reproduction. Any empty cell next to exactly three neighbors is a birth cell. An organism is placed on it in the following generation.

To start the game an initial pattern of live cells is placed in the grid. The colony of cells may grow into a large population, fall into a cyclic pattern, or die off. The complexity and diversity of the life generated in the game exceeded what anyone ever thought the underlying simplistic rules would create.

interesting pattern of cells he termed a
rough four generations returning to its
onally by one cell space. The Life
tarting pattern. It is difficult to see the
gram execution is slow using BASIC. The
compiled or assembly language

ng Explanation

he game of Life is slow. It may take a
een updates. Each screen update

NE (new) are two-dimensional, 24 cells
This two-dimensional array represents
xt screen display that is used. Each
s a cell that may or may not contain an
epresents the current "generation"

places binary 1s in the NE data array.
er.

he program steps through the entire NE
E data array is printed onto the screen. If
ogram prints the space character
chr$(32). If the cell contains a "1" the program prints a block
character: chr$(31).

After a cell is printed onto the screen, the data is transferred from the NE array (new) into the OL array (old). The generation variable GN is printed at the bottom of the screen and incremented by one.

When the screen has been displayed, the computer begins to calculate the next generation. It beings by checking the data in the OL array. Using the data in the OL array, the program creates the NE array.

First the eight neighboring cells are checked for each cell position. For each neighbor found, the variable N is incremented by one. When all neighboring cells have been checked, the outcome of the cell position depends upon the number held in variable N. (See also life rules; survival, death and reproduction)

What isn't stated explicitly in these program lines is the outcome of the cell when N = 2. When N = 2, the cell doesn't change in the next generation. If it contains an organism, the organism makes it to the next generation; if it's empty, it remains empty. The checking algorithm repeats for each cell in the array. Upon completion the program jumps to the display section to display the new generation and the process repeats.

Generations

By replacing the initial glider pattern, we can investigate more of life's little surprises. Replace the original program lines with the following program line to make a new starting pattern. This pattern creates an expanding population of organisms.

```
REM  Enter Starting Pattern
NE(12,31) = 1
NE(12,32) = 1
NE(12,33) = 1
NE(13,30) = 1
```

```
NE(13,33) = 1
NE(14,29) = 1
NE(14,33) = 1
RETURN
```

Experiment with this program by changing the starting patterns as well as modifying the rules. You never know what you may come up with!

3D Life

The game of life has advanced beyond a two-dimensional game. Carter Bays, a computer scientist at the University of Carolina, has created three-dimensional versions of life. One version uses cubes held in three-dimensional space. A cube in *3D Life* has 26 neighbors instead of eight. The other version uses spheres. Unfortunately we don't have the space to explore these programs. Not to worry, for those interested in either version of 3D Life can receive more information by contacting Carter Bays at the University of Carolina.

Carter Bays Computer Science Department
The University of South Carolina
Columbia, SC 29208

Order Out of Chaos (Random Life)

This program is not as complex as the "Game of Life" but it generates very interesting screen images. It is a simple randomly initiated cellular automation. When the program is run, it fills the screen, line by line, with color pixels. When the screen is full, it begins to scroll upwards. Each new line generated on the screen represents a new generation. Pressing the "q" key exits the program, while pressing any other key randomly changes the rules upon which new generations are produced.

The keyboard is checked only once after each new line is printed onto the screen. This makes the program less than responsive to keyboard presses. That is a trade-off to have the BASIC program run as quickly as possible.

Order Out of Chaos—BASIC Listing Explanation

This program uses a fastest algorithm than the Life program. Because of this, we can use pixels instead of chr$() characters, for a higher resolution display. Using pixels, however, slows the BASIC program considerably. I feel this is made up for by the resolution of CA pictures obtained from the program.

The program dimensions four arrays, OL(325) for old array, NE(325) for new array, KO(325) for color and code(16) for code.

The "code" array is filled with 16 binary numbers (1s or 0s). The program randomly changes these numbers upon either request of the

programmer while the program is runnin
scrolls upward. The automatic changing
evolution. The code array may be conside
program. Its sequence determines the nex
see exactly how this happens in a bit.

The user can change the code when
pressing any key except "q." The "q" key

GOSUB "seed" jumps to a subroutin
with random sequence of binary 1s and 0
through the OL array applying the "rule"
generation. Note that the first line of pixe
the seed subroutine, just to get started, a
you calculate all the possible variations i
discover that it simply generates a numbe
binary numbers in the OL array. The num
binary numbers from the previous genera
Y+2 and Y+3. The number produced from
binary number from the CODE array. The
from the code array is placed in the NE a
character is also derived and stored in the

When the entire NE array has been g
the screen.

The next two lines allows the screen

The next three lines accepts keyboar
the program, press "q." Pressing any othe
subroutine "change." This subroutine ran
0s and 1s held in the CODE array.

Modifying the CA Program

As the program is written, it uses a 320 x 200 screen with 16 colors. You can modify the program to us 32 colors or a higher 640 x 400 resolution screen.

The program automatically evolutes by going to the "change" subroutine every time the screen scrolls. You can stop the automatic evolution by changing one line:

```
IF ht = 184 then ht = 170: GOSUB change: LOCATE
23,1:PRIINT:PRINT
```

to

```
IF ht = 184 then ht = 170: LOCATE 23,1:PRINT:PRINT
```

If you change this line, the program will evolve only when the user presses keys as described previously.

The Dark Side of Cellular Automation

The dark side of cellular automation consists of computer worms and viruses. Computer worms and viruses are self-replicating programs that instigate themselves into computer operation systems. Viruses, like their biological counterparts, infect and replicate inside hosts. For computer viruses the hosts are programs files and diskettes.

Viruses are spread from computer to computer through "infected" diskettes or from a computer networks. Viruses that hide in the boot section of the computer's hard disk or floppy diskette are called boot sector viruses. From this vantage point the virus attempts

disks it has the occasion to corrupt.
fame is an example of a boot sector
irectories and file allocation tables
isplaying the message "Your PC is
NA." The file allocation table is a road
e disk telling it where it has stored all

file viruses. These viruses can either
rogram files. If the virus infects a
nto the program itself, generally
tive.
to a program waits until that
itself into memory along with the
n a word processing program with an
tself into the computer memory with
memory, it stays there even when
processor. The virus stays in memory
be run. If you should run another
m, the virus attaches itself to that
s until the virus has infected every
ve.

Worms are a class of viruses. These are usually found on mainframe computers. The most common attack of a worm is to replicate itself ad infinitum until the entire storage system of the computer is full of worm copies. As the worm eats up RAM and disk space, the computer begins to slow down drastically.

Bombs are viruses that are triggered by a date, time, or event. The Jerusalem-B virus goes off every Friday 13. Its main attack is erasing any program run on that day.

Stealth viruses may be either boot or file viruses. It is called a stealth virus because it tries to hide from anti-viral programs. The manner by which the virus hides is ingenious; it changes the disk directory information to conceal its size and location.

Some computer scientist speculated that viruses are the first programs capable of existing without willful cooperation of humans. While this is true, it must be remembered that without humans to originally write the program code, viruses wouldn't exist at all.

Because of the potential damage that can be caused by computer viruses, the problem has created a market for anti-viral programs. There are many commercial anti-viral programs on the market that clean up and prevent viruses from attacking your computer system.

The National Computer Security Association has an eight-page document "Corporate Virus Prevention Policy" for sale. The document is targeted toward individuals who are responsible for their companies anti-virus policy. Cost is $9.95 for a printed copy, $12.95 for a disk with ASCII text file or $19.95 in *WordPerfect* format. Call 717-258-1816 for more information.

Core Wars

Core Wars are a recreational form of computer viruses. The game was created by A.K. Dewdney. The game is simple. Two programs are placed into a computer's memory. Each program's job is to battle and annihilate the other program. The winning program takes over the allocated space in the computer's memory.

The initial publication of the game was in the May 1984 issue of *Scientific American's* "Computer Recreation" column by Dewdney. Since the initial description, the game gained so much popularity that

tournaments are held for the best core war program.

Further information on Core Wars can be received by writing:

The Core War Newsletter
William R. Buckley, Editor
5712 Kern Drive
Huntington Beach CA 92649

International Core War Society
Mark Clarkson
8619 Wassall Street
Wichita, Kans. 67210

Software Available from:

Amiga PCs
Mark A Durham
8282 Cambridge Street # 507
Houston, TX 77054

Free Artificial Life Programs

If you belong to a large BBS network such as CompuServe, it is likely that there are artificial life programs there you may download for free. Check CompuServe's Amiga File Finder.

The classic version of "Life" has multiple listings written with a variety of computer languages. You may want to download a compiled version for faster execution times.

Commercial Artificial Life Programs

We will end this by noting a few commericial artificial life programs. These automation programs are available through standard distribution (stores) or directly from the manufacturer. I haven't personally used the *SimLife* program, but the reviews I have read on it are good.

CellPro
MegageM
1903 Adria
Santa Maria, CA 93454
(805) 349-1104

SimLife
Maxis2 Theater Sq.
Suite 230
Orinda CA 94563
800-336-2947
510-254-9700

✔

The complete set of listings can be found on the AC's TECH disk.

Please write to:
John Iovine
c/o AC's TECH
P.O. Box 2140
Fall River, MA 02722

HUGE NUMBERS

by Michael Griebling

This series of three articles explores an alternate floating point number system (called Extended Precision Numbers or ExNumbers) that allows an almost unlimited number of digits and can exactly represent all decimal fractions.

To demonstrate a practical use of ExNumbers, a scientific calculator is incrementally developed so that each article adds a major new capability. The final calculator has support for logical, transcendental, logarithmic, and exponential operations and is implemented as a Workbench-compatible tool, with either mouse-driven or keypad entry of complete equations.

This article introduces the ExNumbers, describes utilities such as comparison of two numbers, absolute value, digit shifting, and multiplication/division by ten, and develops the basic operations of addition, subtraction, multiplication, and division. Next, various conversions to/from ExNumbers are covered. Finally, there is a brief tour of the initial CLI-based calculator to demonstrate the power of ExNumbers in a real application.

Amiga Math Problems

The Amiga has an almost embarrassing abundance of floating point math libraries that include the Fast Floating Point (FFP) library, IEEE 32-bit and IEEE 64-bit floating point libraries. The last two are supported by the Amiga's floating point coprocessor. But there are serious deficiencies in all of these numerical representations.

Quad Index	Quads	Binary (16-bit integer)
0	1,234	0000 0100 1101 0010
1	5678	0001 0110 0010 1110
2	9000	0010 0011 0010 1000
	⋮	⋮
12	0000	0000 0000 0000 0000
Exponent	+5	0000 0000 0000 0101
Sign	positive	0000 0000 0000 0000

Figure 1: ExNumber representation of the number '123456.789'

The chief problem with these floating point formats is that they are encoded as binary numbers (mathematical base 2) while we tend more naturally to decimal numbers (base 10). As a consequence, the fractional representations of the binary numbers are usually slightly different from the decimal number fractions that we would expect. For example, the number 0.8 is exactly representable as a decimal number while the closest binary fraction, in IEEE 32-bit format, ends up being 0.799999952316284. This difference is very important to business people who can't get their spreadsheets to balance when accumulated round-off errors don't balance out to zero. ExNumbers can represent all decimal fractions exactly so no pennies are gained or lost.

A second problem is that the maximum resolution supported by the IEEE 64-bit format is only about 15 digits. This limitation may be very serious to very large corporations whose accumulated income is numbered in billions of dollars and to space explorers who need to send a space probe to an exact orbital position. With a virtually unlimited number of digits, ExNumbers can again meet the need for higher resolutions.

There is a down side to the use of ExNumbers. Since they are implemented in software, they are slower than coprocessor-based floating point numbers for an equivalent number of digits. Certainly, when using many digits, the calculations are slower. In the proposed calculator application, however, the slower speed is not noticeable.

Structure of ExNumbers

ExNumbers are represented as an array of 16-bit signed words, each of which contains four decimal digits and is called a Quad. The exponent is kept in a separate 16-bit word; and the number's sign is an enumeration of 'positive' and 'negative' values (Figure 1). ExNumber Quads are different from Binary-Coded Decimal (BCD) representations, which are also used to store decimal-encoded numbers. The Quads actually encode four decimal digits using a binary number so they are stored more efficiently and are faster than BCD numbers in calculations.

For example, to encode 123456.789 as an ExNumber, we first need to normalize this number to a value whose mantissa is between

-10 and 10. In other words, the number is represented in scientific notation as 1.23456789E6. Next, this number is broken into groups of four digits, beginning at the leftmost digit. The decomposed number can now be represented as 1234 5678 9000 E0006 where the E0006 is the number's exponent. Note the trailing zeros in the 9000 Quad. These extra digit place holders are required to pad this Quad because, if they were omitted, the final 9 would be interpreted as 0009, which would lead to the erroneous number 123456.780009. The three Quads from this example are stored in binary form as shown in Figure 1.

Utilities

Before dealing with the main mathematical operations, I need to introduce a few utilities that will be used in these algorithms.

The first of these is the ExCompare function, which produces a result to indicate whether one ExNumber is equal to, greater than, or less than another ExNumber. Although the algorithm in Listing 1 for ExCompare appears confusing, Table 1 summarizes the decisions that are used to produce an ExNumber comparison. The notation A(+) indicates that A is positive, Aexp is an abbreviation for A's exponent, and A(i) represents the ith Quad of A.

The ExChgSign procedure negates a number, meaning that the sign is toggled from positive to negative and vice versa.

ExAbs takes the absolute value of a number by forcing the sign to be positive.

ExNorm normalizes an ExNumber by removing leading zeros in a fraction and adjusting the exponent to guarantee a mantissa between -10 and 10. For example, 0.0000456 would be normalized to 4.56E-6.

ExTimes10 increments the exponent by 1 to simulate a multiplication by 10. This procedure is much faster than really multiplying an ExNumber by 10.

Similarly, ExDiv10 subtracts 1 from the exponent to simulate a division by 10. This procedure is also much faster than attempting to divide an ExNumber by 10.

The ExShiftRight procedure is used for shifting a single digit rightmost into an ExNumber's mantissa. Shifting 8 into the number 6.7892 produces the number 8.67892.

ExShiftLeft shifts an ExNumber to the left by a single digit and replaces the least significant digit with a zero.

The IsZero function returns true if the ExNumber argument passed to it is equal to zero.

Basic Arithmetic

Some of us may remember how confusing it was when first learning about binary numbers after having been exposed for most of our lives to decimal numbers. Well, the bad news is that you can forget most of what you learned about binary arithmetic; the good news is that the basic ExNumber operations of addition, subtraction, multiplication, and division are performed in very much the same way that we are used to from our daily lives.

Addition is the most basic operation in the ExNumbers module (Listing 1) and the ExAddUtility is the heart of the addition algorithm that is used by the exported ExAdd procedure. Two positive numbers are added together in three steps: first, set the exponent of the result to the exponent of the larger number; second, shift the smaller number so that its Quads are aligned with the larger number; and finally, add together all the related Quads. The example in

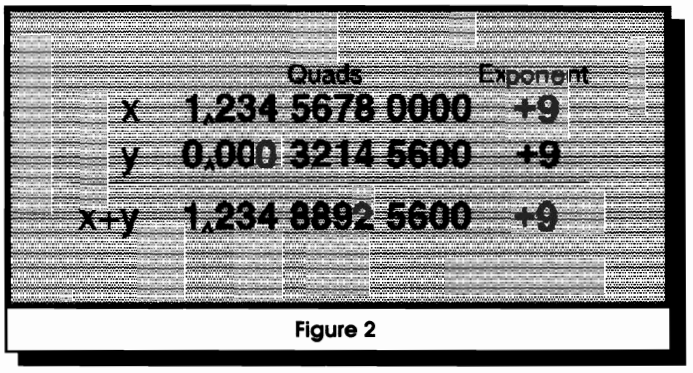


Figure 2

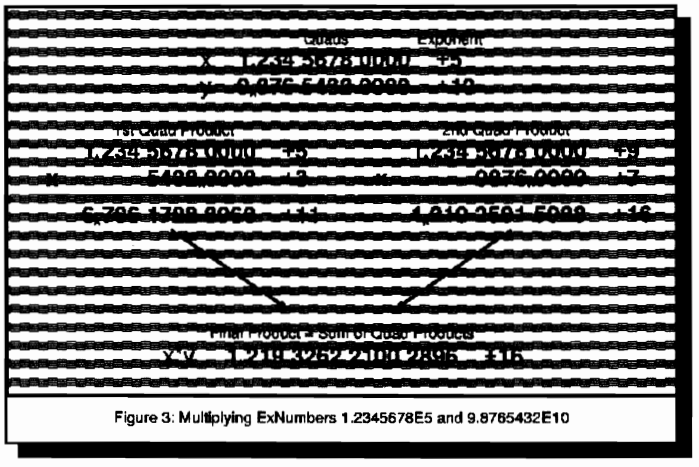


Figure 3: Multiplying ExNumbers 1.2345678E5 and 9.8765432E10

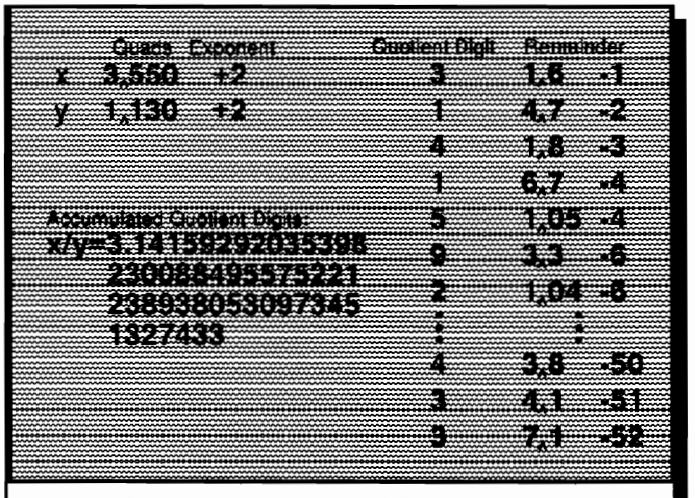


Figure 4: Dividing ExNumber 3.550E2 by 1.130E2

Figure 2 illustrates how the numbers 1.2345678E9 and 3.21456E5 are added together.

The ExSubUtility subtracts two numbers using almost the same steps also used by the ExAddUtility with the exception that Quads are subtracted from each other instead of being added.

To simplify both addition and subtraction algorithms, I made a tacit assumption that both numbers would be positive. The reason this works is seen in how the ExAdd procedure checks and manipulates the signs of the two numbers to be added together. There are

A(i)	B(i)	A=B	A(i)<B(i)	Aexp>Bexp	Aexp=Bexp	Result
T	F	-	-	-	-	A>B
F	T	-	-	-	-	A<B
T	-	-	-	T	F	A>B
F	-	-	-	T	F	A<B
T	-	-	-	F	F	A<B
F	-	-	-	F	F	A>B
-	-	T	-	-	T	A=B
T	-	F	T	-	T	A<B
F	-	F	T	-	T	A>B
T	-	-	F	-	T	A>B
F	-	-	F	-	T	A<B

T — True
F — False
- — Don't Care

Table 1: ExNumber comparison algorithm in tabular form

two possibilities: both numbers have the same sign (either positive or negative) so they can be added together using the ExAddUtility procedure; or the numbers have different signs so we subtract the negative number from the positive number using the ExSubUtility procedure.

The ExSub procedure is even simpler. This algorithm makes use of the well-known property that B - C can be rewritten as B + (-C). Thus, by negating C, a call to the ExAdd procedure produces the correct answer.

ExMult multiplies two ExNumbers together using the same techniques that we were taught in school. Two nested loops produce a product by using the outer loop to index the first number's ith Quad and then the inner loop multiplies this Quad by each of the Quads in the second number to produce an intermediate product. Any carries are then shifted into this intermediate product, the exponent is adjusted accordingly, and the intermediate product is added to the final result. The above process is repeated until an intermediate product has been produced and added to the final result for each Quad in the first number. Figure 3 demonstrates the multiplication of 1.2345678E5 and 9.8765432E10.

The division algorithm should also be familiar to everyone. First, the result's exponent is calculated by subtracting the divisor's exponent from the dividend's exponent. Next, the divisor and dividend are normalized and they are forced to be positive numbers. This step is equivalent to lining up the divisor and dividend prior to beginning a manual division. Once again, two nested loops are used but now the outer loop iterates over all the digits in an ExNumber while the inner loop increments a quotient counter and subtracts the divisor from the dividend as long as the dividend is greater than or equal to the divisor. This is roughly what we do when manually comparing the divisor with the dividend and estimate a quotient that when multiplied by the divisor and subtracted from the dividend leaves the dividend less than the divisor. Our algorithm here, however, replaces the multiplication/subtraction step with just a series of subtractions. Finally, the divisor is divided by 10 to shift it within the range of the dividend for the next digit of the quotient and the whole process repeats. The division algorithm's outer loop guarantees that enough quotient digits are produced to fill all the ExNumber Quads. Figure 4 takes you through the steps involved in dividing 3.550E2 by 1.130E2.

String Conversions

Before getting on to the actual calculator project, a couple of procedures are still missing. We need a way of translating a string into an ExNumber and, conversely, producing a string from an ExNumber.

The StrToExNum procedure accomplishes the first of our goals. This algorithm parses a string into a set of digits (0-9), signs (+, -), and punctuation (.,E). First, leading spaces are stripped off and the sign of the number is determined. Then, as each digit is encountered, it is packed into a Quad at a position just to the right of the previous digit. The ExNumber's Quads are indexed by a digit counter, which is also used to let us know when enough digits have been gathered. An exponent counter is incremented for each digit in the number. When reaching the digit maximum, only the exponent counter continues to be incremented but no more digits are added to the ExNumber. When a decimal is reached, the exponent counter increments are stopped. If an 'E' (exponent) is encountered, the ExNumber's mantissa is considered complete and the exponent's sign is determined. All following digits are merged into the ExNumber's exponent to which the exponent counter is either added or subtracted—depending on whether the exponent is positive or negative.

The second conversion routine, ExNumToStr, produces a character string from an ExNumber. Two different floating point number formats are supported: scientific notation (e.g., 1.23E10) and floating point notation (e.g.,234.234). Floating point notation is used whenever the ExNumber is small enough to be represented in a field of 'MaxDigits', which represents the maximum number of digits selected by the user. Both conversions begin by checking the sign of the ExNumber and inserting a minus sign if the number is negative.

The scientific notation conversion continues by rounding the ExNumber to the number of decimal places specified by the 'Decimal' argument. The leftmost digit is then inserted into the string, followed by a decimal point. Exactly 'Decimal' digits are placed after the decimal point (even if they are all zeros). The exponent symbol, 'E', is added to the output string, followed by the exponent's sign. A Modula-2 library function called ConvNumToStr, which converts integers into strings, is then used to convert the exponent into a string that is appended at the end of the output string.

Converting ExNumbers into floating point notation is a bit more complicated. As before, the number is rounded to the maximum number of digits—in this case, the exponent size plus the number of specified decimal places. If the ExNumber is less than zero, a leading '0.' is placed in the string, followed by enough zeros to reduce the number's exponent to zero. Next, enough digits are placed into the output string to satisfy the requested number of decimal places or exhaust the total number of digits in an ExNumber, whichever comes first. While placing these digits in the output string, a counter (InCnt) also keeps track of the decimal point so it can be placed at the right place in the output string. The last step of the conversion process removes trailing zeros from numbers like 35.123000000 to give more readable numbers like 35.123.

DO YOUR SHOPPING, THEN CALL US!

AMIGA

A4000 Computer	2399
A1200 Computer	395
w/40MB HD Installed	535
w/65MB HD Installed	559
w/85MB HD Installed	619
w/130MB HD Installed	694
w/235MB HD Installed	829
A600 Computer	175
1942 Multisync Monitor	399
1084S Monitor	229
A520 Video Adapter	34
A2088 XT Bridgecard	69
A2091 Hard Drive & Ram Controller w/170MB HD	269
A2091 HD Controller	69
2 MB Ram For 2091	80
Janus 2.1 Update	45
A2000/3000 Disk Drives	69.95
A500 Int. Disk Drives	49.95
A600/1200 Keyboards	29.95
A2000/3000 Keyboard	59.95
A2000/3000 Power Supply	109
External case PS & Cables	99

REMOVABLE A1200/600 HD SYSTEM
- External Box w/Power & Cabling
- QUANTUM 245 HD 2Yr Warranty
- 256K Cache Ram
- As Fast As 7 (SEVEN) ms
- Cable Select Autoconfiguration
- Take Off One System And Plug Into Another W/O Any Hassles

Only $319

AMIGA CUSTOM CHIPS

2.05 Kickstart Rom	39.95
2.04 Kickstart Rom	33.95
1.3 Kickstart Rom	22
1MB Agnus (8372A)	37.50
2MB Agnus (8372B)	89.95
Super Denise (8373)	28.95
Paula (8364) Or Denise	18.95
CIA (8520)	9.50
Gary (5719)	13.95
2620/2630 Upgrade Kit	35
2091 Upgrade Eproms 7.+	35
Superbuster (-11) (4091)	99
Ramsey (A3000)	99
Fat Gary (A3000)	49
Super Dmac (A3000)	99
Amber (A3000)	79
A3000 Daugterboard	99
A3000 Motherboard	199
- comes with 1 mb ram	
- 25mhz CPU w/MMU	
- All Other Sockets Empty	

VIDEO

DPS Personal TBC III	699
DPS Personal TBC IV	849
DPS Personal Animation	1699
DPS Personal Component	449
DPS Personal V Scope	699
Kitchen Sync	1199
Opal Vision 2.0	575
DCTV-NTSC	274
Retina w/2MB	479
Retina w/4MB	549
Vlab-24Bit Digitizer	379
Scala 210MM	299
Deluxe Paint IV AGA	119
Morph Plus	149
Art Department Pro	159
Pixel 3D Professional	149
Video Director	129
Caligari 24	239
Aladdin 4D	249
Video Toaster 3.0 Upgrade	699
Video Toaster 4000	1795
Picaso w/1MB	459

MICROBOTICS

M1230XA ACCELERATOR
68030 RC25Mhz CPU w/MMU & clock
50 MHZ VERSION FOR ONLY 349
Only $249

MBX 1200z COMBO BOARD
68881 RC20Mhz FPU & with clock
NOW ONLY $125

Call For All Other Custom Configurations Available

VXL * 30 ACCELERATOR 68030 & 32 Bit Wide Ram For The A500/2000!

VXL 30: 25 & MMU Mhz $160 | w/68882 25FPU$235 | 40EC Mhz $225 | w/FPU $364 | 33 Mhz & MMU $225 | w/FPU $320 | VXL 32 RAM: 2MB $179 | 8MB $549

UNMATCHED SYQUEST PRICES

44MB DRIVE (SQ555)	$239
88MB (SQ5110C) (R&W44)	$349
105MB IDE	$449
105MB SCSI	$525
44MB Cartriges	$ 65
105MB Cartriges	$ 90
External Versions Add	$ 99

AMIGA 4000 & 3000 OWNERS CRUSH THE 16 MEG BARRIER
- Add up to 128 MB of ram of contiguous memory
- Four simm sockets using industry standard simms
- Create and run animation from ram on the VT4000
- True Zorro III 32 bit memory board
- Play back over 50 seconds of real-time animation in Hi-Res
- Record your animation at a fraction of the cost!

DKB's 3128 is Only $295

GREAT VALLEY PRODUCTS

A500-HD8+0MB/52	249
A500-HD8+0MB/105	299
A500-HD8+0MB/127	325
A500-HD8+0MB/245	399
A530-Turbo/127	535
A530-Turbo/245	599
A530-Turbo+2/127	635
A2000-HC8+0MB	149
SIMM32/1MB/60ns	69.95
SIMM32/4MB/60ns	179
SIMM32/16MB/60NS	1199
1MB SIMM GForce A3000	179
G-Lock Genlock	385
A2000-IV24 Adapter	55

A1230 Turbo+ 40/40/4	$549.00
A1200 SCSI / RAM+0K	$209.00
w/33mhz&4MB Ram	$449.00

*** A530 Special ***
We Won't Be Beat!
GForce Accelerators w/ 68030,68882, 60ns RAM, SCSI Controller, RAM Card in One

40Mhz/4MB	609
50Mhz/4MB	795
68040 33Mhz A2000/4MB	1055

PC286 Module 16Mhz	59
Tahiti-II 1GB (35ms)	2499
Tahiti-II 1GB Cartridge	249
IV 24 Impact Vis. CT	1215
IV 24 Impact Vision 2.0	1049
IV 24 2.0 Upgrade	129
68882 40Mhz FPU PLCC	139
FaaastROM Kit (For HDs)	39
Cinemorph Software	55
Phonepak VFX 2.0	289
DSS8+ Sound Sampler	89
I/O Extender (2SerialPort)	99
Image F/X	249
Image F/X Upgrade	29

ICD

AdIDE 40 MB HD system for Amiga 500 System	220
AdIDE 60 MB HD System for Amiga 500 System	260
Flicker Free Video II	228

DKB

Insider II w/1.5M RAM	180
2632 w/4Megabytes	349
MegAChip 2000/500 w/2MB Agnus	179
Multi-Start 2 Rev 6A	29
KwikStart II for A1000	69
SecurKey Security Board	99
3128 A3000/4000 Ram board Expandable to 128MB w/0K	275

IVS

Grand Slam/500	229/287
Trumpcard Pro/500	139/225
Trumpcard 500 Pro	225
Trumpcard 500 Plus	149
Trumpcard 500 AT	164
Sourcer Switching Power Supply	99
Moviemaker	799

AUDIO

Sunrise Industries

AD516 Digitizer (16Bit)	1189
AD1012 Digitizer (12Bit)	479

EXPANSION SYSTEMS

Baseboard 601C	69
Dataflyer XDS 0MB	75
w/120Maxtor	259
w/245Quantum	339
Baseboard 1200C	25
Baseboard 500 0K	85
Xramboard w/0K	75
Dataflyer Ramcard w 2mb	149
500 SCSI or IDE	125
1000 SCSI or IDE	135
2000 SCSI or IDE	75
500 Express SCSI	165
500 Express IDE	155
500 Express SCSI&IDE	189
500 SCSI&IDE	149
1000 SCSI&IDE	169
2000 SCSI&IDE	95

CSA

DERRINGER 25/25/1 w/MMU,FPU&RAM	399
DERRINGER 50/4 w/MMU,&4MB RAM	599
Rocket Launcher 50/50 w/MMU Makes The CBM 2630 Or GVP Combo 25 run at 50 Mhz w/MMU & FPU (68882RC50)	499

Micro R&D

2000 Bigfoot Power Supply	159
500 Bigfoot Power Supply	89
1200 Bigfoot Power Supply	99
Slingshot A500 (1A2000Slot)	39
Slingshot Pro A500 (Gives An A2000 Slot w/Passthru	69

Hewlett Packard Printers

HP 4SI	3195	DJ1200C	1459
HP 4M	1975	DJ500	319
HP 4	1459	DJ500C	419
HP 4L	675	DJ550C	619
HP 4ML	1079	DJPortable	399
HP IIIP	899	Scanjet IIP	969
		Scanjet IIC	1320

3.5" HARD DRIVES

Warranties
Quantum=2 Years
Maxtor=2 Years

Maxtor 120 LPS SCSI/IDE	179
Maxtor 170 LPS IDE	189
Maxtor 213 LPS SCSI/IDE	229
Maxtor 245 LPS SCSI/IDE	249
Maxtor 345 LPS SCSI/IDE	359
Quantum 52 (Low Profile)	125
Quantum 105 LPS	149
Quantum 127 ELS	179
Quantum 245 LPS SCSI-	2264
Quantum 525 LPS SCSI-	2599
Quantum 1.2 Gig 5 YR	999
Toshiba 1.2 Gig 5 YR	999

2.5" A1200/600 Hard Drives

40M Conner	139
65M Seagate	165
85M Conner / Seagate	215
130M Conner	269
235MB Seagate	439

MEMORY CHIPS

All Speeds Available

1x8 100-60ns SIMMS	32+
4x8 80-60ns SIMMS	129+
1x4 80-60ns Static ZIP	16+
1x4 80-60ns Page ZIP	17+
1x4 80-50ns Page DIP	18+
1x1 120-70ns DIP	5+
256X4 120-60ns DIP	5+
256X4 120-60ns ZIP	5+
256x32 (1MB Simm 72P)	29+
512x32 (2MB Simm 72P)	75+
1x32 Simms	139+
2x32 Simms	299+
4x32 Simms	599+
8x32 Simms	1800+
PCMCIA Ram Card 2M	119+
PCMCIA Ram Card 4M	229+

Ram Changes For Better Or Worse Please Call 1st

MATH CHIPS, CPU's &FPU's

68030-RC-50 w/MMU	179.00
68882-RC-50 (PGA)	149.00
68030-RC-33 w/MMU	129.00
68882-RC-33 (PGA)	95.00
68030-RC-25 w/MMU	99.00
68882-RC-25 (PGA)	75.00
68030-FN-PLCC (Call)	CALL
68882-FN-PLCC (Call)	CALL
80387-25SX (Bridges)	69.95
Crystal Oscillators (All)	10.00

LASER PRINTER MEMORY

HP II, IID, IIP, III, IIID, IIP AND ALL PLUS SERIES	
Board with 2MB	89.00
Board with 4MB	145.00
Deskjet 256K Upgrade	55
HP 4 (4 Meg)	149
HP 4 (8 Meg)	295

ACCESORIES/MISC.

PowerPlayers Joystick	6.49
SupraTurbo 28Mhz	149
Safeskin Protectors	15.00
Xtractor+ Chip Puller	9.95
Kool-It Cooling kit A500	39.95
Qwika Switcha 4 socketed ROM selector	39.95
Power Connectors	CALL
SCSI HD Cables	CALL

386 Bridgecard Owners
80387SX25 Co Processor
Speeds Up Performance
Only $69.95

DeVine COMPUTER SALES

18 Wellington Drive
Newark, DE. 19702
(302) 836-6174 ORDERS ONLY
(302) 836-4145 PRODUCT Info/Tech
(302) 836-8829 Fax 24 HOURS

Please Understand Our Policies

VISA / MASTER Card Accepted. Prices And Specifications Are Subject To Change Without Notice! 15% Restocking Fee On All Returns. Defective Merchandise Will Be Replaced With Same Item. Call 302.836.4145 For Approval RMA# Before Returning Merchandise Merchandise. No Returns After 10 Days From Delivery Date. Not Responsible For Incompatibility Of Products Shipping And Handling For Chips Is $5 COD Fee $6 Personal Checks Require 10 Working Days To Clear. Call For Actual Shipping Prices On All Other Items. Ram Prices Change Daily

The CLI Calculator

Listing 2 gives the complete calculator source code that supports the basic mathematical operations covered in this article. The calculator is implemented with a very basic tokenizer (GetToken), which takes a stream of input characters and translates them into the set of tokens identified at the top of Listing 2 following the 'Tokens' type. Quite a few more tokens are identified than are being used in this first calculator version but this complete list should give a tantalizing view of the features that will be added in the next article.

The stream of tokens drive a simple recursive expression evaluator (Expression) that accepts prioritized infix notation with bracketing limited only by stack size and input string length restrictions (250 characters). Operators are ordered as follows: Highest priority (reciprocal, squared), Medium priority (times, divide), Lowest priority (plus, minus, negate, and unary plus). Thus, an expression like 2 + 5 * 6 would give the expected result of 32, and not 42.

The calculator also supports 'friendly' number entry that allows numbers to contain punctuation characters like commas, apostrophes, and underscores, which can be used to separate groups of digits (e.g., 5,000 and 1'000'000.45). This feature is especially useful with 50-digit numbers!

A Sample Session

To use the calculator, make sure you either are in the directory that contains the calculator program or copy the calculator into a directory that is part of your regular command path. Type 'Calculator' from the CLI and you are greeted with a 'CALC>' prompt. Simply type in the following equation exactly as it appears (hold down the ALT key and press '2' to get the ^{2} character and similarly to get the $^{-1}$ hold the ALT key and press 'N', then '1'), and enter a Return:

```
4 * Pi * (89.234 + 4E1 / 2.0)²≤ - (2⁻¹ * 56.78 * 10)
```

The answer 149658.8731 appears on the next line after the equation. Note the use of the name 'Pi' to represent the mathematical quantity 3.141592..., the squared operator (^{2}), and the reciprocal operator ($^{-1}$). These extensions were trivial to add to the calculator and extend its usefulness. This example also shows a variety of number formats: integer, floating point, and scientific notation.

The default settings for the calculator give floating point number results. If you want to fix the decimal point, just type 'DEC 2', for example, to set the number of decimal points to two digits. To restore to floating point format,type 'DEC 0'. To toggle between floating point and scientific notation type 'SCI'.

Experiment on your own with the calculator to see how it handles various errors like illegal characters and mismatched brackets. If you have a Modula-2 compiler, attempt some extensions to the calculator, to recognize additional mathematical constants like the base of the natural logarithm (e) or attempt a cubed (x^3) operator.

Next Time

In the next article I'll add an ExInteger module that gives us 172-bit logical operations (AND, OR, XOR, BIT, SHIFTs) and base conversion functions. As well, an ExMathLib0 module will be introduced that uses the Amiga's coprocessor to give ExNumber calculations with exponential, power, trigonometric, and logarithmic functions. All these new operators and functions will be integrated into an improved CLI calculator that should be comparable to commercial products.

Listing 1

```
IMPLEMENTATION MODULE ExNumbers;

FROM Conversions IMPORT ConvNumToStr, Dec;
FROM InOut      IMPORT Write, WriteString, WriteLn,
                       WriteLongInt;
FROM Strings    IMPORT LengthStr, AppendSubStr;


CONST
 MaxLengthNumber = 2 * HighBoundsManArray;

VAR
 MaxDigits, MaxQuads : INTEGER;


PROCEDURE SetMaxDigits(D : CARDINAL);
(* Set maximum digits in extended real numbers — must be
   a multiple of 4 *)
BEGIN
  IF D < 4 THEN
    MaxDigits := 4;
    ExStatus := TooFewDigits;
  ELSIF D > HighBoundsManArray THEN
   MaxDigits := HighBoundsManArray;
    ExStatus := TooManyDigits;
  ELSE
    MaxDigits := D DIV 4;   (* Force a multiple of 4 *)
    IF D MOD 4 > 0 THEN INC(MaxDigits) END;
    MaxDigits := MaxDigits * 4;
  END;
  MaxQuads := MaxDigits DIV 4;
END SetMaxDigits;


PROCEDURE GetMaxDigits () : CARDINAL;
(* Get the current number of digits in extended real
numbers *)
BEGIN
 RETURN MaxDigits;
END GetMaxDigits;


PROCEDURE GetExpMant(x : ExNumType; VAR exp : INTEGER;
                     VAR mant : ExNumType);
(* Returned 'mant' number will be between -10.0 and 10.0
*)
BEGIN
  exp := x.Exp;
  mant := x;
  mant.Exp := 0;
END GetExpMant;


PROCEDURE IsZero(A : ExNumType) : BOOLEAN;
VAR
  i : INTEGER;
  Zero : BOOLEAN;
BEGIN
  (* check for zero *)
  i := 0;
```

```
  Zero := TRUE;
  WHILE (i <= MaxQuads) AND Zero DO
    IF A.Man[i] # 0 THEN
      Zero := FALSE;
    END;
    INC(i);
  END;
  RETURN Zero;
END IsZero;


PROCEDURE ExShiftRight(Carry : INTEGER; VAR A :
ExNumType);
(* shift all mantissa digits in A to the right one place.
  The most significant digit is replaced with the Carry.
*)
VAR
  i : INTEGER;
BEGIN
  (* shift right *)
  FOR i := MaxQuads TO 1 BY -1 DO
    A.Man[i] := A.Man[i] DIV 10 + (A.Man[i-1] MOD 10) *
1000;
  END;

  (* put Carry in most significant position *)
  A.Man[0] := A.Man[0] DIV 10 + 1000 * Carry;
END ExShiftRight;


PROCEDURE ExShiftLeft(VAR A : ExNumType);
(* shift all mantissa digits in A to the left one place.
  The least significant digit is replaced with zero. *)
VAR
  i : INTEGER;
BEGIN
  (* shift left *)
  FOR i := -1 TO MaxQuads-1 DO
    A.Man[i] := (A.Man[i] MOD 1000) * 10 + A.Man[i+1] DIV
1000;
  END;

  (* put zero in least significant position *)
  A.Man[MaxQuads] := (A.Man[MaxQuads] MOD 1000) * 10;
END ExShiftLeft;


PROCEDURE ExAddUtility(VAR A : ExNumType; B, C :
ExNumType);
(* A := ABS(B) + ABS(C) *)
VAR
  i, j, joff, carry, quad, total : INTEGER;
  Ex11, Ex2 : ExNumType;
BEGIN
  IF IsZero(B) THEN
    A := C;
  ELSIF IsZero(C) THEN
    A := B;
  ELSE
    IF B.Exp > C.Exp THEN
      Ex11 := B;
      Ex2 := C;
    ELSE
      Ex11 := C;
      Ex2 := B;
    END;
    A := Ex0;
    A.Exp := Ex11.Exp;
    carry := 0;

    (* shift smallest number until quad-aligned relative
to
       larger number *)
    j := (Ex11.Exp - Ex2.Exp) MOD 4;
    FOR i := j TO 1 BY -1 DO
      ExShiftRight(0, Ex2);
      INC(Ex2.Exp);
    END;
    joff := (Ex2.Exp - Ex11.Exp) DIV 4;

    (* add the two numbers together *)
    FOR i := MaxQuads TO 0 BY -1 DO
      (* j = index to Ex2 *)
      j := i + joff;

      (* check that j falls within array bounds *)
      IF (j >= 0) AND (j <= MaxQuads) THEN
        (* get quad digit from Ex2 *)
        quad := Ex2.Man[j];
      ELSE
        (* j is outside array bounds, use 0 for quad
digit *)
        quad := 0;
      END;

      (* perform addition with carry *)
      total := Ex11.Man[i] + quad + carry;

      (* check for carry *)
      IF total >= 10000 THEN
        DEC(total, 10000);
        carry := 1;
      ELSE
        carry := 0;
      END;
      A.Man[i] := total;
    END;

    (* handle final carry *)
    IF carry = 1 THEN
      (* shift carry into top of mantissa *)
      ExShiftRight(carry, A);

      (* multiply by ten to update exponent *)
      ExTimes10(A);
    END;
  END;

  (* set ExStatus *)
  IF A.Exp > MaxExp THEN
    ExStatus := Overflow;
  END;
END ExAddUtility;


PROCEDURE ExSubUtility(VAR A : ExNumType; B, C :
ExNumType);
(* A := ABS(B) - ABS(C) *)
VAR
  PositiveResult : BOOLEAN;
  i, j, joff, borrow, quad, result : INTEGER;
  Ex11, Ex2 : ExNumType;
BEGIN
  ExAbs(B);
  ExAbs(C);
  IF IsZero(B) THEN
    A := C;
  ELSIF IsZero(C) THEN
    A := B;
  ELSE
    IF B.Exp > C.Exp THEN
      Ex11 := B;
```

```
      Ex2 := C;
    ELSE
      Ex11 := C;
      Ex2 := B;
    END;
   PositiveResult := ExCompare(Ex11, Ex2) = ExGreater;
    A := Ex0;
    A.Exp := Ex11.Exp;
    borrow := 0;

   (* shift smallest number until quad-aligned relative
to
       larger number *)
    j := (Ex11.Exp - Ex2.Exp) MOD 4;
    FOR i := j TO 1 BY -1 DO
     ExShiftRight(0, Ex2);
     INC(Ex2.Exp);
    END;
    joff := (Ex2.Exp - Ex11.Exp) DIV 4;

    (* subtract the two numbers *)
    FOR i := MaxQuads TO 0 BY -1 DO
      (* j = index to Ex2 *)
      j := i + joff;

      (* check that j falls within array bounds *)
      IF (j >= 0) AND (j <= MaxQuads) THEN
        (* get quad from Ex2 *)
       quad := Ex2.Man[j];
      ELSE
       (* j is outside array bounds, use 0 for quad *)
        quad := 0;
      END;

      (* perform subtraction with borrow *)
      IF PositiveResult THEN
       result := Ex11.Man[i] - quad - borrow;
      ELSE
       result := quad - Ex11.Man[i] - borrow;
      END;

      (* check for borrow *)
      IF result < 0 THEN
       INC(result, 10000);
        borrow := 1;
      ELSE
        borrow := 0;
      END;
     A.Man[i] := result;
    END;
  END;

  (* normalise *)
  ExNorm(A);

  (* adjust sign *)
  IF ExCompare(B, C) = ExLess THEN
   ExChgSign(A);
  END;
END ExSubUtility;


PROCEDURE ExAdd(VAR A : ExNumType; B, C : ExNumType);
(* A = B + C *)
BEGIN
  IF B.Sign = C.Sign THEN
    (* B and C have the same sign — just add *)
   ExAddUtility(A, B, C);
   IF B.Sign = negative THEN
     ExChgSign(A);
    END;
  ELSE
    (* B and C have different signs *)
   IF B.Sign = positive THEN
    ExSubUtility(A, B, C);
   ELSE
    ExSubUtility(A, C, B);
   END;
  END;
END ExAdd;


PROCEDURE ExSub(VAR A : ExNumType; B, C : ExNumType);
(* A = B - C *)
BEGIN
  ExChgSign(C);   (* A = B + (-C) *)
  ExAdd(A, B, C);
END ExSub;


PROCEDURE ExMult(VAR A : ExNumType; B, C : ExNumType);
(* Return B * C *)
VAR
  i, j, carry : INTEGER;
  product : LONGINT;
  Ex1 : ExNumType;
BEGIN
 IF (ExCompare(B,Ex0) = ExEqual) OR (ExCompare(C,Ex0) =
ExEqual) THEN
    (* multiplication by zero *)
    A := Ex0;
 ELSIF ExCompare(C,Ex1) = ExEqual THEN
    A := B;
 ELSIF ExCompare(B,Ex1) = ExEqual THEN
    A := C;
  ELSE
    (* real multiplication *)
    A := Ex0;
    FOR i := MaxQuads TO 0 BY -1 DO
      Ex1 := Ex0;
      Ex1.Exp := B.Exp + C.Exp - i * 4 - 3;
      carry := 0;
      FOR j := MaxQuads TO 0 BY -1 DO
      product := LONGINT(B.Man[j]) * LONGINT(C.Man[i])
+ LONGINT(carry);
       Ex1.Man[j] := product MOD 10000;
       carry := product DIV 10000;
      END;

     (* check for final carry *)
     WHILE carry > 0 DO
      ExShiftRight(carry MOD 10, Ex1);
       ExTimes10(Ex1);
       carry := carry DIV 10;
      END;

     (* perform summation *)
     ExAddUtility(A, A, Ex1);
    END;

   (* adjust product sign *)
   IF B.Sign # C.Sign THEN
     ExChgSign(A);
    END;
  END;
END ExMult;


PROCEDURE ExDiv(VAR A : ExNumType; B, C : ExNumType);
(* A := B / C *)
VAR
  i, j : INTEGER;
  quotient : LONGINT;
```

```
  Ex11, Ex2 : ExNumType;
BEGIN
  IF IsZero(C) THEN
    (* attempt to divide by zero *)
   ExStatus := DivideByZero;
  ELSIF IsZero(B) THEN
    (* dividend = 0 *)
    A := Ex0;
  ELSIF ExCompare(C,Ex1) = ExEqual THEN
    (* divisor = 1 *)
    A := B;
  ELSE
    (* real division *)
    A := Ex0;
   A.Exp := B.Exp - C.Exp;

   (* adjust quotient sign *)
   IF B.Sign # C.Sign THEN
     ExChgSign(A);
    END;

   (* let Ex11 = ABS(B) / magnitude of B *)
    Ex11 := B;
   ExAbs(Ex11);
    Ex11.Exp := 0;

   (* let Ex2 = ABS(C) / magnitude of C *)
    Ex2 := C;
   ExAbs(Ex2);
    Ex2.Exp := 0;

   (* actual division *)
   FOR i := 0 TO MaxDigits-1 DO
     quotient := 0;
    WHILE ExCompare(Ex11, Ex2) >= ExEqual DO
       INC(quotient);
      ExSubUtility(Ex11, Ex11, Ex2);
     END;
    A.Man[i DIV 4] := A.Man[i DIV 4] * 10 +
INTEGER(quotient);
    ExDiv10(Ex2);
   END;

   (* normalize quotient *)
   ExNorm(A);
  END;
END ExDiv;


PROCEDURE ExChgSign(VAR A : ExNumType);
(* A := -A *)
BEGIN
  IF A.Sign = positive THEN
    A.Sign := negative;
  ELSE
    A.Sign := positive;
  END;
END ExChgSign;


PROCEDURE ExAbs(VAR A : ExNumType);
(* A := ABS(A) *)
BEGIN
 A.Sign := positive;
END ExAbs;


PROCEDURE ExRound(VAR A : ExNumType; D : CARDINAL);
(* A := Round(A) *)
VAR
 cindex, index, digit, i : INTEGER;
 Ex1 : ExNumType;
```



```
BEGIN
 IF D <= CARDINAL(MaxDigits-1) THEN
   index := (D+1) DIV 4;
  digit := A.Man[index];
   cindex := (D + 1) MOD 4;
   IF cindex = 0 THEN
     digit := digit DIV 1000;
   ELSIF cindex = 1 THEN
     digit := digit DIV 100;
   ELSIF cindex = 2 THEN
     digit := digit DIV 10;
   END;
   IF digit MOD 10 >= 5 THEN
     (* round up *)
     Ex1 := Ex1;
    Ex1.Exp := A.Exp - INTEGER(D);
    IF A.Sign = negative THEN
      ExChgSign(Ex1);
     END;
     ExAdd(A, A, Ex1);
   END;

   (* make remaining digits zero *)
   IF cindex = 0 THEN
     A.Man[index] := 0;
   ELSIF cindex = 1 THEN
    A.Man[index] := A.Man[index] DIV 1000 * 1000;
   ELSIF cindex = 2 THEN
    A.Man[index] := A.Man[index] DIV 100 * 100;
   ELSIF cindex = 3 THEN
    A.Man[index] := A.Man[index] DIV 10 * 10;
   END;
   FOR i := index+1 TO MaxQuads DO
     A.Man[i] := 0;
   END;
 END;
END ExRound;


PROCEDURE PutDigit(VAR A : INTEGER; Digit, Index :
INTEGER);
BEGIN
  IF Index = 0 THEN
    A := A MOD 1000 + Digit * 1000;
```

```
  ELSIF Index = 1 THEN
    A := A DIV 1000 * 1000 + A MOD 100 + Digit * 100;
  ELSIF Index = 2 THEN
    A := A DIV 100 * 100 + A MOD 10 + Digit * 10;
  ELSE
    A := A DIV 10 * 10 + Digit;
  END;
END PutDigit;


PROCEDURE ExTrunc(VAR A : ExNumType);
(* Truncate A so no decimal places are kept. *)
VAR
  i : INTEGER;
BEGIN
  FOR i := A.Exp+1 TO MaxDigits-1 DO (* zero these digits
*)
    PutDigit(A.Man[i DIV 4], 0, i MOD 4);
  END;
END ExTrunc;


PROCEDURE ExFrac(VAR A : ExNumType);
(* Keep only the fraction portion of A. *)
VAR
  i : INTEGER;
BEGIN
  FOR i := 0 TO A.Exp DO (* zero these digits *)
    PutDigit(A.Man[i DIV 4], 0, i MOD 4);
  END;
  ExNorm(A);              (* normalize the fraction *)
END ExFrac;


PROCEDURE ExToLongCard(A : ExNumType) : LONGCARD;
(* Convert the extended real number 'A' into a cardinal -
   saturating if necessary. *)
CONST
  MaxDigits = 10;
VAR
  Cnt : INTEGER;
  Int : LONGCARD;
BEGIN
  IF A.Exp < 0 THEN
    RETURN 0;
  ELSIF A.Exp >= MaxDigits THEN
    RETURN MAX(LONGCARD);
  ELSE
    Int := 0;
    FOR Cnt := 0 TO A.Exp DO
      A.Man[-1] := 0;
      ExShiftLeft(A);
      IF Cnt = MaxDigits-1 THEN
        IF Int > MAX(LONGCARD) DIV 10 THEN
          RETURN Int;
        END;
        IF (Int = MAX(LONGCARD) DIV 10) & (A.Man[-1] > 6)
THEN
          A.Man[-1] := 6;
        END;
      END;
      Int := Int * 10 + LONGCARD(A.Man[-1]);
    END;
    RETURN Int;
  END;
END ExToLongCard;


PROCEDURE ExToLongInt(A : ExNumType) : LONGINT;
(* Convert the extended real number 'A' into an integer -
   truncating if necessary. *)
VAR
  Int : LONGCARD;
  Negative : BOOLEAN;
BEGIN
  Negative := FALSE;
  IF A.Sign = negative THEN
    Negative := TRUE;
    ExAbs(A);
  END;
  Int := ExToLongCard(A);
  IF Int > MAX(LONGINT) THEN Int := MAX(LONGINT) END;
  IF Negative THEN
    RETURN -LONGINT(Int);
  ELSE
    RETURN LONGINT(Int);
  END;
END ExToLongInt;


PROCEDURE ExCompare(A, B : ExNumType) : ExCompareType;
(* Compares the two extended real numbers. *)
VAR
  Done : BOOLEAN;
  i : INTEGER;
BEGIN
  IF A.Sign # B.Sign THEN
    (* A and B have different signs *)
    IF A.Sign = positive THEN
      (* A and B have different signs and A is positive
so A>B *)
      RETURN ExGreater;
    ELSE
      (* A and B have different signs and A is negative
so A<B *)
      RETURN ExLess;
    END;
  ELSE
    (* A and B have the same sign *)
    IF (A.Exp # B.Exp) & NOT IsZero(B) & NOT IsZero(A)
THEN
      IF A.Exp > B.Exp THEN
        (* A exponent > B exponent *)
        IF A.Sign = positive THEN
          RETURN ExGreater;
        ELSE
          RETURN ExLess;
        END;
      ELSE
        (* A exponent <= B exponent *)
        IF A.Sign = positive THEN
          RETURN ExLess;
        ELSE
          RETURN ExGreater;
        END;
      END;
    ELSE
      (* A & B have same sign and A exponent = B exponent
*)
      Done := FALSE;
      i := 0;

      (* compare each digit until a difference is found
or
         we reach the end *)
      WHILE (i <= MaxQuads) AND NOT Done DO
        IF A.Man[i] # B.Man[i] THEN
          Done := TRUE;
        ELSE
          INC(i);
        END;
      END;
      IF i > MaxQuads THEN
```

```
      (* end reached and all digits match *)
      RETURN ExEqual;
     ELSE
      (* compare different digits *)
      IF A.Man[i] < B.Man[i] THEN
        IF A.Sign = positive THEN
          RETURN ExLess;
        ELSE
          RETURN ExGreater;
        END;
       ELSE
        IF A.Sign = positive THEN
          RETURN ExGreater;
        ELSE
          RETURN ExLess;
        END;
       END;
     END;
   END;
  END;
END ExCompare;


PROCEDURE ExMin(VAR A : ExNumType; B, C : ExNumType);
(* Return the smaller of B and C in A *)
BEGIN
 IF ExCompare(B, C) = ExGreater THEN
   A := C;
 ELSE
   A := B;
 END;
END ExMin;


PROCEDURE ExMax(VAR A : ExNumType; B, C : ExNumType);
(* Return the larger of B and C in A *)
BEGIN
 IF ExCompare(B, C) = ExLess THEN
   A := C;
 ELSE
   A := B;
 END;
END ExMax;


PROCEDURE ExTimes10(VAR A : ExNumType);
(* A := A * 10 — much faster than ExMult *)
BEGIN
 INC(A.Exp);
 IF A.Exp > MaxExp THEN
   ExStatus := Overflow;
 END;
END ExTimes10;


PROCEDURE ExDiv10(VAR A : ExNumType);
(* A := A / 10 — much faster than ExDiv *)
BEGIN
 DEC(A.Exp);
 IF A.Exp < MinExp THEN
   ExStatus := Underflow;
 END;
END ExDiv10;


PROCEDURE ExNorm(VAR A : ExNumType);
(* Normalise A *)
BEGIN
 (* normalise *)
 IF IsZero(A) THEN
   (* normalize zero *)
```



```
   A.Sign := positive;
   A.Exp := 0;
 ELSE
  (* shift mantissa to left until most significant
digit is
     non-zero, increment exponent with each shift *)
   WHILE A.Man[0] DIV 1000 = 0 DO
     ExShiftLeft(A);
     ExDiv10(A);
   END;
 END;
END ExNorm;


PROCEDURE WriteExNum(A : ExNumType;
                     Width, Decimal, ExpWidth : CARDI-
NAL);
(* Write out A to the current output stream in a field of
  'Width' characters, with 'Decimal' decimal places, and
  'ExpWidth' exponent width. *)
VAR
 S : ARRAY [0..MaxLengthNumber] OF CHAR;
 i, len : CARDINAL;
BEGIN
 ExNumToStr(A, Decimal, ExpWidth, S);
 len := LengthStr(S);
 IF Width >= len THEN
   FOR i := 1 TO Width-len DO Write(" ") END;
 END;
 WriteString(S);
END WriteExNum;


(*$S-*)
PROCEDURE StrToExNum(S : ARRAY OF CHAR; VAR A :
ExNumType);
(* Convert the string 'S' into an extended real number in
A. *)
VAR
 Exp, NumbIndex, InCnt, EndCnt : INTEGER;
 ZeroFlag, NegativeExponent, LeftSide, InExponent :
BOOLEAN;
 Done, NegExponent : BOOLEAN;
 ActiveChar : CHAR;
```

```
 PROCEDURE SetDigit(VAR Numb : INTEGER);
 BEGIN
   Numb := Numb * 10 + INTEGER(ORD(S[InCnt]) -
ORD('0'));
 END SetDigit;

BEGIN
  (* initialize a few counters and stuff *)
  A := Ex0;
  InCnt := 0;              (* character counter *)
  Exp := 0;                (* working exponent *)
 LeftSide := TRUE;
 InExponent := FALSE;
 ZeroFlag := TRUE;
 NegativeExponent := FALSE;
 EndCnt := LengthStr(S);
 NumbIndex := 0;

  (* set the sign of 'A' to a negative - if needed *)
 WHILE (InCnt < EndCnt) & (S[InCnt] = ' ') DO INC(InCnt)
END;
 IF S[InCnt] = '-' THEN
   A.Sign := negative;
   INC(InCnt);
 END;
 WHILE InCnt < EndCnt DO
  ActiveChar := S[InCnt];
  IF (ActiveChar >= '0') & (ActiveChar <= '9') THEN
    IF InExponent THEN
      SetDigit(Exp);
     ELSE
      IF NumbIndex < MaxDigits THEN  (* enter a digit
*)
       SetDigit(A.Man[NumbIndex DIV 4]);
       END;
      IF ZeroFlag & (S[InCnt] # '0') THEN
        ZeroFlag := FALSE;
       END;
      IF NOT ZeroFlag THEN
        INC(NumbIndex);
       IF LeftSide THEN INC(A.Exp) END;
       ELSE
        IF NOT LeftSide & (A.Exp <= 0) THEN DEC(A.Exp)
END;
       END;
     END;
  ELSIF ActiveChar = '.' THEN
    IF ~LeftSide THEN ExStatus := IllegalNumber END;
     LeftSide := FALSE;
  ELSIF ActiveChar = 'E' THEN
    InExponent := TRUE;
    IF S[InCnt+1] = '-' THEN
      NegativeExponent := TRUE;
       INC(InCnt);
    ELSIF S[InCnt+1] = '+' THEN
      INC(InCnt);
     END;
   ELSE
    ExStatus := IllegalNumber;
  END; (* IF *)
  INC(InCnt);
 END;

 (* fix up the last quad digits *)
 WHILE (NumbIndex DIV 4 <= MaxQuads) & (NumbIndex MOD 4
> 0) DO
  A.Man[NumbIndex DIV 4] := A.Man[NumbIndex DIV 4] *
10;
  INC(NumbIndex);
 END;

 (* Do some final fixes to the exponent *)
 IF NegativeExponent THEN
   DEC(A.Exp, Exp);
 ELSE
   INC(A.Exp, Exp);
 END;
 DEC(A.Exp);

 (* Ensure valid zero value *)
 IF IsZero(A) THEN A := Ex0 END;
END StrToExNum;


PROCEDURE ExNumToStr(A : ExNumType; Decimal, ExpWidth :
CARDINAL;
                     VAR S : ARRAY OF CHAR);
(* Convert the extended real number into a string 'S'. *)
VAR
 pos, ManIndex, StrCnt, InCnt : INTEGER;
 ExpStr : ARRAY [0..40] OF CHAR;
 FixPoint, Ok : BOOLEAN;

 PROCEDURE ConcatChar(ch : CHAR);
 BEGIN
   S[pos] := ch;
   INC(pos);
 END ConcatChar;

 PROCEDURE GetDigit() : CHAR;
 BEGIN
   INC(StrCnt);
   IF StrCnt = 4 THEN (* get a quad of digits *)
     Ok :=
ConvNumToStr(ExpStr,LONGCARD(A.Man[ManIndex]),Dec,FALSE,4,'0');
    INC(ManIndex);
    StrCnt := 0;
   END;
  RETURN ExpStr[StrCnt];
 END GetDigit;

BEGIN
 (* initialize a few parameters *)
 pos := 0;
 StrCnt := 3;
 ManIndex := 0;
 ExpStr := '';

 (* force scientific notation for numbers too small or
too large *)
 IF (ExpWidth = 0) AND (ABS(A.Exp) > MaxDigits) THEN
  ExpWidth := 1;  (* force scientific notation *)
 END;

 (* add the negative sign to the number *)
 IF A.Sign = negative THEN ConcatChar('-') END;

 (* ensure we don't exceed the maximum digits *)
 FixPoint := Decimal # 0;
 IF (INTEGER(Decimal) > MaxDigits) OR NOT FixPoint THEN
  Decimal := MaxDigits-1;
 END;

 (* convert the number into scientific notation *)
 IF ExpWidth > 0 THEN
  ExRound(A, Decimal);    (* round to appropriate
decimal places *)
  ConcatChar(GetDigit()); (* leading digit *)
   ConcatChar('.');       (* decimal point *)
   FOR InCnt := 1 TO INTEGER(Decimal) DO
    ConcatChar(GetDigit());     (* add following digits
*)
   END;
```

```
    (* add the exponent *)
   ConcatChar('E');
   IF A.Exp >= 0 THEN ConcatChar('+') END;(* this looks
better *)
    ConcatChar(0C);                          (* terminate
the string *)
    Ok :=
ConvNumToStr(ExpStr,LONGINT(A.Exp),Dec,TRUE,ExpWidth,'0');
   AppendSubStr(S, ExpStr);
  ELSE
   (* format a non-scientific number *)
   ExRound(A, INTEGER(Decimal)+A.Exp); (* round to
decimal places *)
    IF A.Exp < 0 THEN
      ConcatChar('0');                  (* leading digit
*)
      ConcatChar('.');                  (* decimal point
*)
     FOR InCnt := 2 TO ABS(A.Exp) DO   (* pad with
leading zeros *)
       ConcatChar('0');
      END;
     INC(Decimal, A.Exp+1);
    END;
    InCnt := 0;
    REPEAT
     ConcatChar(GetDigit());
      IF InCnt > A.Exp THEN
        DEC(Decimal);
      ELSIF InCnt = A.Exp THEN
        ConcatChar('.');
      END;
      INC(InCnt);
   UNTIL (InCnt = MaxDigits) OR (Decimal = 0);
   ConcatChar(0C);

   (* remove any trailing zeros and unneeded digits *)
    InCnt := pos - 2;
   WHILE (InCnt > 1) & (S[InCnt] = '0') & NOT FixPoint
DO
     S[InCnt] := 0C;
     DEC(InCnt);
    END;
  END;
END ExNumToStr;


PROCEDURE ExNumb(LeftMan : LONGINT; RightMan : LONGCARD;
                 ExpShift : INTEGER; VAR A : ExNumType);
(* create an extended real number which has LeftMan to
the left
   of the decimal point and RightMan to the right. The
ExpShift
   quantity can shift the decimal point to the right for
negative
   values; to the left for positive values. *)
VAR
  i : INTEGER;
BEGIN
  A := Ex0;
  IF LeftMan < 0 THEN
    A.Sign := negative;
    LeftMan := -LeftMan;
  END;
  WHILE RightMan # 0 DO
   ExShiftRight(RightMan MOD 10, A);(* shift right 1
position *)
    RightMan := RightMan DIV 10;
  END;
  WHILE LeftMan # 0 DO
   ExShiftRight(LeftMan MOD 10, A); (* shift right 1
position *)
    ExTimes10(A);                   (* adjust the
exponent *)
    LeftMan := LeftMan DIV 10;
  END;
  ExDiv10(A);                       (* final exponent
adjust *)
  INC(A.Exp, ExpShift);             (* shift the decimal
point *)
  IF A.Exp > MaxExp THEN            (* signal any errors
*)
    ExStatus := Overflow;
  ELSIF A.Exp < MinExp THEN
    ExStatus := Underflow;
  END;
END ExNumb;


BEGIN
  (* create extended number 0 *)
  Ex0.Sign := positive;
  FOR MaxDigits := -1 TO HighBoundsManArray DIV 4 DO
    Ex0.Man[MaxDigits] := 0;
  END;
  Ex0.Exp := 0;

  (* default to max number of digits *)
  SetMaxDigits(HighBoundsManArray);

  (* create some extended number constants *)
  ExNumb(1, 0, 0, Ex1);     (* 1.0 *)

  StrToExNum(
  "3.14159265358979323846264338327950288419716939937511",
pi);
  StrToExNum(
  "2.71828182845904523536028747135266249775724709369996",
e);
  StrToExNum(
  "0.69314718055994530941723212145817656807550013436026",
ln2);
  StrToExNum(
  "2.30258509299404568401799145468436420760110148862877",
ln10);
END ExNumbers.
```

✔

The complete set of listings can be found on the AC's TECH disk.

Please write to:
Michael Griebling
c/o AC's TECH
P.O. Box 2140
Fall River, MA 02722

PasteUp

by J. T. Steichen

PasteUp is a collection of text file editing commands that operate on the entire file, such as a global search and replace function for a given word contained in the file. Initially, I was going to simply add an ARexx function host to a *CanDo* graphical-user-interface that would call the editing functions that were already written in C. After thinking about it for awhile though, I realized that CanDo was powerful enough to perform the editing functions without going through ARexx. The only drawback was a loss of speed, since CanDo scripting commands are probably interpreted. In order to make it more than a toy, I added ARexx calls to PasteUp and attached them to the different cards (which is what CanDo calls the different screens and windows in an application) so that the user of PasteUp can call the available editing functions from an ARexx program. There is an example ARexx program that shows how to use all of the features available from PasteUp called 'TestPasteUp.rexx'.

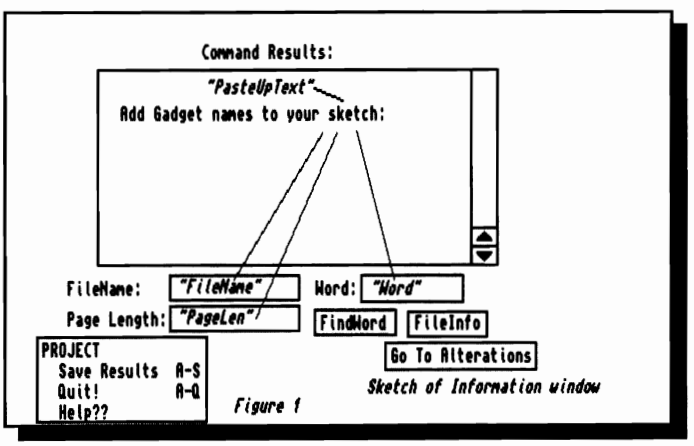


Figure 1

For this article, I decided to show people the method that's needed to create an application using CanDo.

Seven Steps

Step one: know what you want to do. In this case, I wanted to take commands I've used to cut and paste C header and code files together and give them a graphical-user-interface and an ARexx interface. Those commands are:

```
FindWord word_to_find filename_to_search
FileInfo filename [page_size]
Remove starting_line_# ending_line_# input_filename       [output_filename]
Insert after_line_# insert_filename destination_file [output_filename
Extract starting_line_# ending_line_# input_filename      [output_filename]
Delete word_to_delete input_filename [output_filename]
Replace old_word new_word input_filename [output_filename]
```

I realize knowing what you want to do is easier said than done, but a little bit of planning goes a long way in getting a CanDo application working correctly (and looking good!) with a minimum amount of editing. PasteUp has only four cards in its CanDo deck, but you can imagine what kind of headaches you can make for yourself if the deck had 20 or more cards that share data, you find a problem on one, change it, and then have to change several more when they use the same data or a similar method of interfacing with the user of your

application! This rapidly degenerates into a half-working mess that says "rewrite me!" when all you had to do was a little planning up front.

Step two: split the task of the application into logical segments, segments that can be placed in different windows. In the case of PasteUp, these are

TitleScreen
Information source
Alterations
Help information

FindWord and FileInfo ended up in the Information window, since they don't alter any files that are examined. The rest of the editing commands are in the Alteration window.

Step three: Figure out what the user has to input and what has to be returned by each command. For example, the FindWord command requires the user to enter a word to find and a filename to find it in. FindWord returns the line numbers and the lines in which the word was found, grouped together into a list gadget. This means that the Information card has to have at least four gadgets on it: a button gadget to activate the FindWord function, a string gadget for the search word, a string gadget for the filename and a list gadget for the results. I also decided that the user might want to save the results, so I added a 'Save Results' menu item. The FindWord button not only activates the function, but also has to check and make sure that the user gave a search word and filename. For those of you that are interested in how to do this in a CanDo script, I dumped the contents of the PasteUp deck into a file using a CanDo utility called 'ThePrinter.' I edited out information that describes colors and locations of objects, since it's irrelevant.

As you can see, a description of how to go about creating an application is not cut and dried! One trick I have learned in dealing with CanDo applications is shown in Figure 1. I make a rough sketch of how I want the graphical-user-interface to look, label the string and list gadgets with the names that CanDo will use in its scripting language and then start putting it together. Initially, the objects don't have to do anything except look right on the screen. Once everything is placed correctly, then you can start to determine the behavior of each button, string gadget, and list gadget, and whether you want menus, etc.

Step four: Test the behavior of each object and verify that it is doing only what it is supposed to do. CanDo objects are not necessarily easy to debug; for example, the buttons and list gadgets can have four different scripts attached to them, depending on what you want them to do and how the user activates them with the mouse (clicking, dragging, double-clicking and releasing the mouse button over them!). I wish I had kept a log of how I incrementally added buttons and list behaviors while I developed PasteUp in order to give you a better idea how sticking to a logical system of development made this one of the easier decks I've written for CanDo, but this article will probably end up being too long as it is!

Step five: Determine any refinements that need to be made. In the case of PasteUp, I determined that the ARexx interface should always require an output filename for those commands that alter a file. This way, the original input file won't be altered in an undesirable manner. Removing lines from a file is not something that can be easily undone!

Step six: Document the behavior that is not intuitively obvious to the user. For instance, it was easy to add a method to place numbers in the Start # and End # Integer gadgets by simply clicking on the input list gadget in the Alteration window. This is not something that leaps out at you when you look at the Alteration window!

Step seven: Document everything else! The last thing I did to PasteUp was to add comments to all of the object scripts and the ARexx scripts, so that you can figure out why a script is doing what it's doing.

The only ToolType in the icon is ARexxPortName. PasteUp's ARexx port name can be changed to anything you like; the default is 'PasteUp_ARexxPort'. The ARexx commands that PasteUp recognizes are fully documented in the help file and in the example ARexx program 'TestPasteUp.rexx'.

Miscellaneous Notes

You can quit PasteUp without saving the results of your work, so if you want the results, save them before quitting! The ARexx command interface doesn't have this problem.

The help system has a searching problem that you can work around by simply clicking on the first word in the index, or by moving the cursor of the help text gadget to the top of the text. Searches will then work properly. Is this a bug in CanDo V2.01?

PasteUp.prt contains all of the scripts for every button, string gadget, window and list gadget. You will notice that CanDo scripts are very readable. Comments start with a semicolon within the scripts. I hope these scripts inspire you to purchase CanDo; it's great for making application programs without the hassle of trying to figure out how to make complicated C programs. I have the SAS Institute C compiler V6.0 for the Amiga; it's a good product, but Commodore supplies the Amiga Header files that the compiler uses. This means that programs are more difficult to write for the Amiga than they should be.

These are the files that you should have with PasteUp and a description of their contents:

PasteUp	The executable program.
PasteUp.small	The executable which requires the CanDo library.
PasterUp.help	The PasteUp help system.
PasteUp.iff	The title screen
PasteUp index	The index for the PasteUp help system
PasteUp.prt	Listing of the CanDo deck contents for PasteUp
TestPasteUp.rexx	Example ARexx program that uses PasteUp.

All of the files that begin with PasteUp, except for PasteUp.prt, are necessary and should be located in the same directory, in order for PasteUp to work properly. The inspiration for the C commands that PasteUp was written to replace are from "The C Library" by Kris Jamsa.

TestPasteUp.rexx

```
/
*************************************************************************
** TestPasteUp.rexx checks to see that all the ARexx
commands available
** through PasteUp are working properly.
**
** Check all of the *Results.txt files that
TestPasteUp.rexx produces
** in order to see how the ARexx commands work!
*************************************************************************/

Options FailAt 5

if ~show( 'l', "rexxsupport.library" ) then do
  check = addlib( 'rexxsupport.library',0,-30,0)
end

ADDRESS "PasteUp_ARexxPort"  /* Default name for
PasteUp's port! */

'GotoInformation' /* move from the TitleScreen to the
Information window. */

Options Results

/* FileInfo usage:
** FileInfo File_Name page_size
*/

'FileInfo TestPasteUp.rexx 60' /* Count lines, words,
characters & pages */
Say 'FileInfo = ' result

/* Result string format:
** # lines: xx, # words: xx, # char's: xx, # pages: xx
*/

/* FindWord usage:
** FindWord word_to_find File_Name
*/

'FindWord TestPasteUp.rexx TestPasteUp.rexx'

/* The Results of FindWord will be written into
FindWord.txt in the
** following format:
**   Line #: Text from File_Name that contains
word_to_find
*/

'GotoAlteration' /* change to Alteration window of
PasteUp. */

/* Delete usage:
** Delete word_to_delete File_Name outfilename
*/

'Delete ThisWord TestPasteUp.rexx DeleteResults.txt'

/* after TestPasteUp.rexx is run, add the word you'd like
to delete from
** the file back to the Delete command call!
*/

/* Extract usage:
** Extract StartLine_# End_# File_Name OutputFileName
*/

'Extract 63 65 TestPasteUp.rexx ExtractResults.txt'

/* These lines will be found in ExtractResults.txt after
TestPasteUp.rexx
** is run
*/

/* Remove usage:
** Remove StartLine_# EndLine_# File_Name OutputFileName
*/

'Remove 63 65 TestPasteUp.rexx RemoveResults.txt'

/* Insert usage:
** Insert StartLine_# InsertFile Destination_File
OutputFileName
*/

'Insert 63 ExtractResults.txt RemoveResults.txt
InsertResults.txt'

'NoDisplay'   /* Push PasteUp display to back of all
open screens. */

Do d = 1 to 900
   Nop
End

/* Replace usage:
** Replace Old_Word New_Word File_Name OutputFileName
*/

'Replace ReplaceResult TestPasteUp.rexx TestPasteUp.rexx
ReplaceResults.txt'

'Quit'  /* ShutDown PasteUp program! */
```

PasteUp.prt

```
********************************
* Deck "PasteUp"
* Date 07/24/93
********************************

********************************
* Card(s) in deck.
*  Card "Alteration"
*   Card "Help"
*  Card "Information"
*  Card "TitleCard"
********************************
*  4 Card(s), 4 were printed.
********************************

********************************
```

```
* Natural order of Cards
* Card "TitleCard"
* Card "Information"
*  Card "Alteration"
*  Card "Help"
*******************************

*******************************
* Global Routine(s) in deck.
* Routine "SetupColors"
*******************************

*******************************
* Card "Alteration"

  Routine "DeleteWord"
     Local count                        ;DeleteWord,word
SubRoutine.
     Let count = NumberOfChars( Arg1 )
     Delete CHARACTER, count
   EndScript

  AfterAttachment ; used to be AfterStartup
     Do "SetupColors"
     Global GotStartLine
     Global GotEndLine
      SetPen 4
    PrintText "Command Results:",200,12 ;Used for
development only!
     PrintText "Input  File:",15,100
     PrintText "Output File:",15,112
     PrintText "Source Text:",217,123
      SetPen 2
     PrintText "Start #:",396,100
      PrintText "End   #:",396,112
      SetPen 12
     PrintText "Old Text:",564,101
     PrintText "New Text:",564,124
      Let GotStartLine = FALSE
      Let GotEndLine   = FALSE
   EndScript

  Routine "DeleteWords"
     Local title    ;DeleteWords SubRoutine.
     Local numlines  ;Called by ARexxDelete command.
      Local filename
      Local cntr
      Local safety
      Let title    = WindowTitle
     Let filename = TextFrom( "InFileName" )
    LoadDocument filename,"SourceFile"
     Let numlines = LinesInDocument
    SetWindowTitle "Deleting "||TextFrom( "OldText"
)||" from "||filename||"..."
     WorkWithDocument "ResultText"
      Clear DOCUMENT
     WorkWithDocument "SourceFile"
     MoveCursorTo STARTOF DOCUMENT
       SearchFor TextFrom( "OldText" ),BYWORD
       Do "DeleteWord",TextFrom( "OldText" )
      Let safety = TheLineNumber
       While safety <= numlines
        WorkWithDocument "SourceFile"
        SearchFor TextFrom( "OldText" ),BYWORD
           If SearchFound = TRUE
              Do "DeleteWord",TextFrom( "OldText" )
            Let safety = TheLineNumber
           EndIf
          Let safety = safety + 1
      EndLoop
     SetWindowTitle "Copying results..."
     WorkWithDocument "SourceFile"
     MoveCursorTo STARTOF DOCUMENT
     Let numlines = LinesInDocument
      While numlines > 0
         Let line = TheLine
         MoveCursor DOWN
        WorkWithDocument "ResultText"
         Type line,NEWLINE
        WorkWithDocument "SourceFile"
         Let numlines = numlines - 1
      EndLoop
    LoadDocument filename,"SourceFile"
     WorkWithDocument "SourceFile"
     MoveCursorTo STARTOF DOCUMENT
     Let filename = TextFrom( "OutFileName" )
    SetWindowTitle "Save "||filename||"..."
    SaveDocument "ResultText",filename,ASCII  ;Save
results in outfile!
     WorkWithDocument "ResultText"           ;Update
displays:
     MoveCursorTo STARTOF DOCUMENT
     WorkWithDocument "SourceFile"
     MoveCursorTo STARTOF DOCUMENT
      SetWindowTitle title
   EndScript

  Routine "ExtractLines"
     Local title    ;ExtractLines SubRoutine.
     Local numlines  ;Called by ARexxExtract command.
      Local filename
      Local cntr
      Local line
      Let title    = WindowTitle
     Let filename = TextFrom( "InFileName" )
    SetWindowTitle "Extracting lines "||IntegerFrom(
"StartLine" )||" to "||IntegerFrom( "EndLine" )||"..."
     WorkWithDocument "ResultText"
      Clear DOCUMENT
     WorkWithDocument "SourceFile"
     MoveCursorTo STARTOF DOCUMENT
     Let numlines = LinesInDocument
      Let cntr     = 1
     While cntr < IntegerFrom( "StartLine" )
         Let line = TheLine
         MoveCursor DOWN
         Let cntr = cntr + 1
      EndLoop
     WorkWithDocument "SourceFile"
     While cntr <= IntegerFrom( "EndLine" )
          Let line = TheLine
          MoveCursor DOWN
        WorkWithDocument "ResultText"
```

```
        Type line,NEWLINE
       Let cntr = cntr + 1
      WorkWithDocument "SourceFile"
      EndLoop
      Let filename = TextFrom( "OutFileName" )
   SetWindowTitle "Save "||filename||"..."
   SaveDocument "ResultText",filename,ASCII ;Save
results in outfile!
    WorkWithDocument "ResultText"          ;Update
displays:
    MoveCursorTo STARTOF DOCUMENT
    WorkWithDocument "SourceFile"
    MoveCursorTo STARTOF DOCUMENT
    SetWindowTitle title
  EndScript

  Routine "InsertLines"
    Local title    ;InsertLines SubRoutine.
    Local numlines ;Called by ARexxInsert command.
    Local filename
    Local insfilename
    Local cntr
    Local line
    Let title      = WindowTitle
    Let filename   = TextFrom( "InFileName" )
    Let insfilename = Arg1
   SetWindowTitle "Inserting at line # "||IntegerFrom(
"StartLine")||"..."
    WorkWithDocument "ResultText"
    Clear DOCUMENT
    WorkWithDocument "SourceFile"
    MoveCursorTo STARTOF DOCUMENT
    Let cntr = 1
    While cntr < (IntegerFrom( "StartLine" ) + 1)
       Let line = TheLine
       MoveCursor DOWN
      WorkWithDocument "ResultText"
       Type line,NEWLINE
       Let cntr = cntr + 1
      WorkWithDocument "SourceFile"
     EndLoop
   LoadDocument insfilename,"Temp"
    MoveCursorTo STARTOF DOCUMENT
    SetWindowTitle "Inserting file
"||insfilename||"..."
    Let numlines = LinesinDocument
    While numlines > 0
       Let line = TheLine
       MoveCursor DOWN
      WorkWithDocument "ResultText"
       Type line,NEWLINE
       Let numlines = numlines - 1
      WorkWithDocument "Temp"
     EndLoop
   SetWindowTitle "Appending..."
   WorkWithDocument "SourceFile"
    Let numlines = TheLineNumber
    Let cntr     = LinesInDocument
    While numlines <= cntr
       Let line = TheLine
       MoveCursor DOWN
        WorkWithDocument "ResultText"
       Type line,NEWLINE
       Let numlines = numlines + 1
        WorkWithDocument "SourceFile"
      EndLoop
    Let filename = TextFrom( "OutFileName" )
   SetWindowTitle "Saving "||filename||"..."
   SaveDocument "ResultText",filename,ASCII ;Save
results in outfile!
    WorkWithDocument "ResultText"          ;Update
displays:
    MoveCursorTo STARTOF DOCUMENT
    WorkWithDocument "SourceFile"
    MoveCursorTo STARTOF DOCUMENT
    SetWindowTitle title
  EndScript

  Routine "RemoveLines"
    Local title    ;RemoveLines SubRoutine.
    Local numlines ;Called by ARexxRemove command.
    Local filename
    Local cntr
    Local line
    Let title = WindowTitle
    Let filename = TextFrom( "InFileName" )
   SetWindowTitle "Removing lines "||IntegerFrom(
"StartLine")||" to "||IntegerFrom( "EndLine")||"..."
    WorkWithDocument "ResultText"
    Clear DOCUMENT
    WorkWithDocument "SourceFile"
    MoveCursorTo STARTOF DOCUMENT
    Let numlines = LinesInDocument
    Let cntr     = 1
    While cntr < IntegerFrom( "StartLine" )
       Let line = TheLine
       MoveCursor DOWN
      WorkWithDocument "ResultText"
       Type line,NEWLINE
       Let cntr = cntr + 1
      WorkWithDocument "SourceFile"
     EndLoop
    WorkWithDocument "SourceFile"
    While cntr <= IntegerFrom( "EndLine" )
       MoveCursor DOWN
       Let cntr = cntr + 1
     EndLoop
    Let numlines = numlines - cntr
    Let line     = TheLine
    While numlines > 0
      WorkWithDocument "SourceFile"
       Let line     = TheLine
       MoveCursor DOWN
      WorkWithDocument "ResultText"
       Type line,NEWLINE
       Let numlines = numlines - 1
     EndLoop
    Let filename = TextFrom( "OutFileName" )
   SetWindowTitle "Save "||filename||"..."
   SaveDocument "ResultText",filename,ASCII ;Save
results in outfile!
    WorkWithDocument "ResultText"          ;Update
```

```
displays:
    MoveCursorTo STARTOF DOCUMENT
    WorkWithDocument "SourceFile"
      MoveCursorTo STARTOF DOCUMENT
      SetWindowTitle title
  EndScript

  Routine "ReplaceWords"
    Local title    ;ReplaceWords SubRoutine.
    Local filename  ;Called by ARexxReplace command.
     Local cntr
     Local numlines
     Let title    = WindowTitle
    Let filename = TextFrom( "InFileName" )
   LoadDocument filename,"SourceFile"
   SetWindowTitle "Replacing "||TextFrom( "OldText"
)||" with "||TextFrom( "NewText" )||"..."
    WorkWithDocument "SourceFile"
    MoveCursorTo STARTOF DOCUMENT
   Replace TextFrom( "OldText" ),TextFrom( "NewText"
),BYWORD GLOBAL
    WorkWithDocument "ResultText"
     Clear DOCUMENT
    SetWindowTitle "Copying results..."
    WorkWithDocument "SourceFile"
    MoveCursorTo STARTOF DOCUMENT
    Let numlines = LinesInDocument
     While numlines > 0
        Let line = TheLine
        MoveCursor DOWN
       WorkWithDocument "ResultText"
        Type line,NEWLINE
       WorkWithDocument "SourceFile"
        Let numlines = numlines - 1
     EndLoop
   LoadDocument filename,"SourceFile"
    Let filename = TextFrom( "OutFileName" )
   SetWindowTitle "Save "||filename||"..."
   SaveDocument "ResultText",filename,ASCII  ;Save
results in outfile!
     WorkWithDocument "ResultText"           ;Update
displays:
    MoveCursorTo STARTOF DOCUMENT
    WorkWithDocument "SourceFile"
    MoveCursorTo STARTOF DOCUMENT
    SetWindowTitle title
  EndScript

  Window "UserWindow"
     Definition
       Title "PasteUp - Alterations:"

     OnActivated
        Do "SetupColors"
        SetPen 4
       PrintText "Command Results:",200,12
        PrintText "Input  File:",15,100
        PrintText "Output File:",15,112
        PrintText "Source Text:",217,123
         SetPen 2
        PrintText "Start #:",396,100
         PrintText "End   #:",396,112
        SetPen 12
       PrintText "Old Text:",564,101
         PrintText "New Text:",564,124
         Let GotStartLine = FALSE
        Let GotEndLine   = FALSE
     EndScript
  EndObject

  TextButton "Insert"
     Definition
        Text " INSERT! "

     OnRelease
       Local title    ;Insert Button Release event
script.
        Local numlines
        Local filename
        Local insfilename
        Local cntr
        Local line
        Let title    = WindowTitle
       Let filename = TextFrom( "InFileName" )
        If filename  = ""
          Let filename = AskForFileName(
TheCurrentDirectory, "Which file??", 10, 10 )
        EndIf
        If NOT Exists( filename )
          Let filename = AskForFileName(
TheCurrentDirectory, "Which file??", 10, 10 )
        EndIf
        If filename = ""
         SetWindowTitle "No Source filename given!!"
          Delay 0,4,0
          SetWindowTitle title
          ExitScript
        EndIf
       If IntegerFrom( "StartLine" ) <= 0  ;Simple
error check!
         SetWindowTitle "Start # has to be greater
than 0!!"
          Delay 0,4,0
          SetWindowTitle title
          ExitScript
        EndIf
       Let insfilename = AskForFileName(
TheCurrentDirectory, "Which file to insert??", 10, 10 )
        If insfilename = ""
          ExitScript  ;User selected cancel on file
requester!
        EndIf
       SetText "InFileName",filename
       SetWindowTitle "Inserting at line #
"||IntegerFrom( "StartLine" )||"..."
       WorkWithDocument "ResultText"
        Clear DOCUMENT
       WorkWithDocument "SourceFile"
       MoveCursorTo STARTOF DOCUMENT
         Let cntr     = 1
       While cntr < (IntegerFrom( "StartLine" ) + 1)
;Skip to insert point:
          Let line = TheLine
```

```
            MoveCursor DOWN
          WorkWithDocument "ResultText"
           Type line,NEWLINE
             Let cntr = cntr + 1
             WorkWithDocument "SourceFile"
         EndLoop
       LoadDocument insfilename,"Temp"
       MoveCursorTo STARTOF DOCUMENT
       SetWindowTitle "Inserting file
"||insfilename||"..."
       Let numlines = LinesinDocument
       While numlines > 0
          Let line = TheLine
          MoveCursor DOWN
         WorkWithDocument "ResultText"
          Type line,NEWLINE              ;Write insert
file line by line.
          Let numlines = numlines - 1
         WorkWithDocument "Temp"
        EndLoop
       SetWindowTitle "Appending..."
       WorkWithDocument "SourceFile"
       Let numlines = TheLineNumber
        Let cntr     = LinesInDocument
       While numlines <= cntr   ;add rest of destina-
tion file:
          Let line = TheLine
          MoveCursor DOWN
         WorkWithDocument "ResultText"
          Type line,NEWLINE
          Let numlines = numlines + 1
         WorkWithDocument "SourceFile"
        EndLoop
      WorkWithDocument "ResultText"  ;update displays:
       MoveCursorTo STARTOF DOCUMENT
       WorkWithDocument "SourceFile"
       MoveCursorTo STARTOF DOCUMENT
        SetWindowTitle title
    EndScript
  EndObject

  TextButton "Remove"
     Definition
        Text " REMOVE! "

     OnRelease
       Local title    ;Remove Button Release event
script.
        Local numlines
        Local filename
        Local cntr
        Local line
        Let title    = WindowTitle
       Let filename = TextFrom( "InFileName" )
        If filename  = ""
          Let filename = AskForFileName(
TheCurrentDirectory, "Which file??", 10, 10 )
         EndIf
        If NOT Exists( filename )
          Let filename = AskForFileName(
TheCurrentDirectory, "Which file??", 10, 10 )
         EndIf
        If filename = ""
         SetWindowTitle "No Source filename given!!"
            Delay 0,4,0
            SetWindowTitle title
          ExitScript
        EndIf
       If IntegerFrom( "StartLine" ) > IntegerFrom(
"EndLine" )
         SetWindowTitle "End # has to be greater than
Start #!!"
          Delay 0,4,0
         SetWindowTitle title  ;Simple error check!
          ExitScript
        EndIf
       SetText "InFileName",filename
      SetWindowTitle "Removing lines "||IntegerFrom(
"StartLine")||" to "||IntegerFrom( "EndLine")||"..."
       WorkWithDocument "ResultText"
        Clear DOCUMENT
       WorkWithDocument "SourceFile"
       MoveCursorTo STARTOF DOCUMENT
       Let numlines = LinesInDocument
        Let cntr     = 1
       While cntr < IntegerFrom( "StartLine" ) ;Move
unaffected lines
          Let line = TheLine                ;to
ResultText:
          MoveCursor DOWN
         WorkWithDocument "ResultText"
          Type line,NEWLINE
          Let cntr = cntr + 1
         WorkWithDocument "SourceFile"
        EndLoop
       WorkWithDocument "SourceFile"
       While cntr <= IntegerFrom( "EndLine" ) ;Skip
over lines to be
          MoveCursor DOWN                ;removed:
          Let cntr = cntr + 1
        EndLoop
       Let numlines = numlines - cntr
        Let line     = TheLine
       While numlines > 0
         WorkWithDocument "SourceFile"  ;Append end of
file to results:
          Let line     = TheLine
          MoveCursor DOWN
         WorkWithDocument "ResultText"
          Type line,NEWLINE
          Let numlines = numlines - 1
        EndLoop
      WorkWithDocument "ResultText"  ;Update dis-
plays:
       MoveCursorTo STARTOF DOCUMENT
      WorkWithDocument "SourceFile"
       MoveCursorTo STARTOF DOCUMENT
        SetWindowTitle title
    EndScript
  EndObject

  Memo "PasteUpAlter"
```

```
      Definition
       Document "ResultText" ; where the text comes
from
   EndObject

  TextField "InFileName"
     Definition
       InitialText "Source_File"

     OnClick
       Local filename         ;Get filename from user
& load it:
        Local title
        Let title = WindowTitle
        SetText "InFileName",""
        Let filename = AskForFileName(
TheCurrentDirectory, "Source File...", 10, 12 )
        If filename <> ""
          SetText "InFileName",filename
         EndIf
        If Exists( filename )
          LoadDocument filename,"SourceFile"
          MoveCursorTo STARTOF DOCUMENT
         Else
          SetWindowTitle "file "||filename|||"doesn't
exist!!"
           Delay 0,5,0
          SetWindowTitle title
         EndIf
     EndScript
   EndObject

  TextField "OutFileName"
     Definition
       InitialText "Destination_File"

     OnClick
       Local filename   ;Get a new filename from user:
        Local title
        Let title = WindowTitle
        SetText "OutFileName",""
        Let filename = AskForFileName(
TheCurrentDirectory, "Destination File...", 10, 12 )
        If filename <> ""
          SetText "OutFileName",filename
         EndIf
        If Exists( filename )
         SetWindowTitle "File "||filename|||"already
exists!!"
           Delay 0,5,0
          SetWindowTitle title
          SetText "OutFileName",""
         Else
          SetText "OutFileName",filename
         EndIf
     EndScript
   EndObject

  TextButton "Extract"
     Definition
       Text " EXTRACT! "

     OnRelease
       Local title    ;Extract Button Release event
script.
         Local numlines
        Local filename
        Local cntr
        Local line
        Let title    = WindowTitle
       Let filename = TextFrom( "InFileName" )
        If filename  = ""
          Let filename = AskForFileName(
TheCurrentDirectory, "Which file??", 10, 10 )
        EndIf
       If NOT Exists( filename )
          Let filename = AskForFileName(
TheCurrentDirectory, "Which file??", 10, 10 )
        EndIf
        If filename = ""
         SetWindowTitle "No Source filename given!!"
           Delay 0,4,0
          SetWindowTitle title
           ExitScript
        EndIf
       If IntegerFrom( "StartLine" ) > IntegerFrom(
"EndLine" )
         SetWindowTitle "End # has to be greater than
Start #!!"
           Delay 0,4,0
          SetWindowTitle title
           ExitScript
        EndIf
       SetText "InFileName",filename
      SetWindowTitle "Extracting lines "||IntegerFrom(
"StartLine" )||" to "||IntegerFrom( "EndLine" )||"..."
       WorkWithDocument "ResultText"
        Clear DOCUMENT
       WorkWithDocument "SourceFile"
       MoveCursorTo STARTOF DOCUMENT
       Let numlines = LinesInDocument
        Let cntr     = 1
       While cntr < IntegerFrom( "StartLine" )
          Let line = TheLine  ;skip to start of
extraction point:
           MoveCursor DOWN
           Let cntr = cntr + 1
         EndLoop
       WorkWithDocument "SourceFile"
       While cntr <= IntegerFrom( "EndLine" )
          Let line = TheLine
          MoveCursor DOWN
         WorkWithDocument "ResultText"
          Type line,NEWLINE    ;write extraction lines
to ResultText.
           Let cntr = cntr + 1
         WorkWithDocument "SourceFile"
         EndLoop
      WorkWithDocument "ResultText"  ;update displays:
       MoveCursorTo STARTOF DOCUMENT
       WorkWithDocument "SourceFile"
       MoveCursorTo STARTOF DOCUMENT
```

```
        SetWindowTitle title
    EndScript
  EndObject

  TextButton "Delete"
    Definition
      Text " DELETE! "

    OnRelease
      Local title    ;Delete Button Release event
script.
       Local numlines
       Local filename
       Local cntr
       Local safety
       Let title    = WindowTitle
      Let filename = TextFrom( "InFileName" )
       If filename  = ""
         Let filename = AskForFileName(
TheCurrentDirectory, "Which file??", 10, 10 )
       EndIf
       If NOT Exists( filename )
         Let filename = AskForFileName(
TheCurrentDirectory, "Which file??", 10, 10 )
       EndIf
       If filename = ""
        SetWindowTitle "No Source filename given!!"
         Delay 0,4,0
        SetWindowTitle title
         ExitScript
       EndIf
      If TextFrom( "OldText" ) = ""
        SetWindowTitle "Enter word to delete in Old
Text Gadget!!"
         Delay 0,4,0
        SetWindowTitle title  ;User messed up!
         ExitScript
       EndIf
     SetText "InFileName",filename
     LoadDocument filename,"SourceFile"
      Let numlines = LinesInDocument
     SetWindowTitle "Deleting "||TextFrom( "OldText"
)||" from "||filename||"..."
     WorkWithDocument "ResultText"
      Clear DOCUMENT
     WorkWithDocument "SourceFile"
     MoveCursorTo STARTOF DOCUMENT
     SearchFor TextFrom( "OldText" ),BYWORD
     Do "DeleteWord",TextFrom( "OldText" ) ;There's
no 'Delete Word' command!
      Let safety = TheLineNumber          ;Prevent
infinite loop!
      While safety <= numlines
        WorkWithDocument "SourceFile"
        SearchFor TextFrom( "OldText" ),BYWORD
         If SearchFound = TRUE        ;Delete next
occurance:
           Do "DeleteWord",TextFrom( "OldText" )
           Let safety = TheLineNumber
         EndIf
        Let safety = safety + 1
      EndLoop
        SetWindowTitle "Copying results..."
     WorkWithDocument "SourceFile"
     MoveCursorTo STARTOF DOCUMENT
        Let numlines = LinesInDocument
        While numlines > 0
         Let line = TheLine
         MoveCursor DOWN
        WorkWithDocument "ResultText"
         Type line,NEWLINE
        WorkWithDocument "SourceFile"
         Let numlines = numlines - 1
       EndLoop
      WorkWithDocument "ResultText"     ;Update
displays:
      MoveCursorTo STARTOF DOCUMENT
     LoadDocument filename,"SourceFile" ;Reload
Source File!
      WorkWithDocument "SourceFile"
      MoveCursorTo STARTOF DOCUMENT
       SetWindowTitle title
    EndScript
  EndObject

  TextButton "Replace"
    Definition
      Text " REPLACE! "

    OnRelease
      Local title    ;Replace Button Release event
script.
       Local filename
```

✔

The complete set of listings can be found on the AC's TECH disk.

Please write to:
J.T. Steichen
c/o AC's TECH
P.O. Box 2140
Fall River, MA 02722

GaugeClass

A Class of Gauge Images for Intuition

by Scott Palmateer

Recently I was working on one of my many programming projects that never seem to get finished, and I realized that it might be handy to include a "gauge" display somewhere in the window to keep the user informed as to the progress of whatever function was currently being worked on. Many programs use this type of display. For instance, a 3-D renderer can put up a gauge to let the user know the "percent completed" of the image. The Amiga's operating system itself used a gauge, in version 1.3 and earlier, to display the "percent filled" of a disk volume. There are many possible uses, and since I thought that I might need this same functionality in other projects, I decided to learn about object-oriented programming (OOP) on the Amiga, and GaugeClass was born.

Objects and Classes

Just what *is* OOP? More than anything else, it is a philosophy of programming. Most people think of the C++ programming language when they think of OOP, and C++ definitely does support OOP, but OOP is not limited to one or two languages. It is the organization of data and functions that makes an application "object-oriented." The program that is the focus of this article, gaugeclass.c, is written in the C language.

OOP, like the more familiar "modular programming" style, urges the programmer to think of the programming task at hand as a task that can be broken down into several smaller tasks. By working on these smaller tasks one at a time, the programmer can maintain a great deal of control over how the program develops and operates. OOP, however, employs a different concept, a hierarchy of classes of objects, to make this modularity even more pronounced. The classic analogy used in explaining this is to compare the hierarchy of OOP classes to the hierarchy of classes in biology (Figure 1). For instance, the "animal" class in biology describes things that move, breathe, etc., while the "plant" class includes things that are green and use photosynthesis to make their food. Both of these classes, though, are sub-classes of the class of "living things," and so they therefore share certain features or attributes, one of which is the quality of being alive. In addition, both of these classes may be broken down into sub-classes themselves (e.g. "land animals," "sea animals," "one-celled plants," "trees," etc.). The sub-classes share features with their parent class, but the addition or modification of the parent class's qualities is what differentiates them from their parent class. OOP has a hierarchy of classes much like this, made up of classes, sub-classes, and super-classes. In the jargon of OOP, a class inherits the attributes of its parent class, or superclass, and passes them on to its subclasses. A class can also add attributes to further distinguish it from the superclass. What is an OOP object then?

In OOP, an object is an "instance" of a particular class. What this means is that the object is a unique record of that class's attributes. To return to the biology analogy, Rusty and Rover are both dogs (members of the "dog" class), but they are different in terms of weight, breed, etc. The thing to remember is that they are still both dogs, and so certain actions (walking, feeding) can apply to both. In OOP, these actions correspond to functions operating on data, and are called on object's "methods." The OOP philosophy encourages us to think that since all objects of a particular class are similar in some way, it should be possible to have a set of "methods" ready and waiting for them. An object's class gathers up all these methods in one central place so that they can be re-used for all objects of that class.

This is where one of the main benefits of OOP comes in to play. Since we have a hierarchy of classes, and qualities are inherited from class to class, why not methods too? For instance, we said earlier that the class of "animals" included those things that breathe. Since all land animals breathe, and the class of "land animals" is a sub-class of the class of "animals," we simply inherit the "breathing" method from the superclass, "animals." In the programming world, this feature allows for the sharing of a whole lot of code. Programmers need no longer write a "print_structureA" function for structure A and a separate "print_structureB" function for structure B; if both structure A and B are objects of classes that share a common superclass, they can share the superclass's "print" function.

The final analogy to make between OOP and biology is the concept of polymorphism. What this basically means is that an object can act as a member of its superclass(es). For instance, a dog is a land animal, so we can use any of the "land animal" class methods on it, such as "run." Since land animals are also animals, then a dog is also "polymorphically" a member of the class of "animals." In the programming world, polymorphism has been around for a while, at least in the Amiga's operating system. Just look at Exec Messages and Nodes. Any structure that includes an Exec Message at the start, like an ARexx message, can be considered an Exec Message. Therefore, any of the Exec Message "methods" (PutMsg(), GetMsg()) can be called on it, and they will work. The same goes for Exec Nodes; any

structure beginning with a Node can be considered, from Exec's point of view, a genuine Node. Starting with Release 2 of the operating system, Intuition got object-oriented, and incorporated BOOPSI, which stands for Basic Object-Oriented Programming System for Intuition.

BOOPSI

Aside from the playful name, BOOPSI is a wonderful addition to the operating system. It really makes programming a whole lot easier and faster when it comes to programming in Intuition. BOOPSI provides a class hierarchy of images and gadgets. No longer does the programmer have to allocate a bunch of structures and set them up just so. Now all you need is one call to a new Intuition function called NewObject(). This function will attempt to make the requested object, and will return a pointer to the object if successful. When you are done with your object, just call DisposeObject(). These two functions are really methods defined at the "rootclass" level (the class of which all other classes are subclasses), and in keeping with the concept of polymorphism, any object can be created and destroyed using these functions (see figure 2). BOOPSI also allows the programmer to construct images and gadgets that interact without the programmer having to do any coding. For instance, a proportional gadget can automatically update a string gadget without the programmer having to monitor the gadgets at all! An example of this is given in the *AMIGA ROM Kernel Reference Manual: Libraries*, and one of my supplied programs does something similar. BOOPSI also allows the programmer to create brand new classes as subclasses of existing classes, and that is what I have done with GaugeClass.

As mentioned above, "rootclass" is the class of which all classes are subclasses. There are three immediate subclasses: "imageclass" (the class of images), "gadgetclass" (the class of gadgets), and "icclass" (the class of interconnection class objects, used to connect objects together). I am only going to describe "rootclass" and "imageclass" in this article, since these are the classes of which GaugeClass is a subclass.

There are several methods defined at the "rootclass" level. These methods are those that every class is going to need, like how to create a new object, how to destroy one no longer needed, how to set an object's attributes, and how to get an object's attributes. Through the vehicle of polymorphism, all subclasses of "rootclass" inherit these methods, and therefore do not need to make their own creation/deletion/setting/getting functions.

The "imageclass" class defines its own methods that concern images, like drawing and erasing them. It comes with four subclasses, "frameiclass" (used to make border images), "sysiclass" (used to render system images), "itexticlass" (text images), and "fillrectclass" (filled rectangles). You may be wondering why all these classes are grouped together, since they seem very different. It used to be the case that to render an image required one function, to render text required another one, and borders required yet another. BOOPSI provides a solution in polymorphism. Since text, borders, and images are now considered "imageclass" objects, they can now all be rendered with the same function, DrawImage(). The internals of these classes may differ greatly, and indeed they do, but the programmer need not concern himself with these details. The programmer can let the operating system decide how to render these structures. Like I said, OOP makes programming much easier!

GaugeClass

As I said earlier, I needed some image that graphically reflected some fraction of completion. After considering the subclasses of "imageclass" I decided to make "gaugeclass" a subclass of "imageclass," because it seemed to belong there. This was an arbitrary decision, really, but I think it makes sense, based on how I wanted the gauge to look. It seemed to have so much of its own identity that it did not seem to belong to any of the existing subclasses of "imageclass." Hierarchies can be kind of fuzzy at times. Just ask any biologist!

I decided that my class should provide objects that were very easy to use. I wanted to allow the programmer to choose the colors of the gauge, whether or not it had a frame around it, and of course the programmer should be able to set the fractional value of the gauge. I kept the gauge as simple as possible, in order to make it easy to use for the programmer, and also because I was learning about BOOPSI. This simplicity comes at the cost of some flexibility (e.g. the user cannot control the appearance of the gauge's indicator rectangle), but for most purposes, I think the cost was worth the ease of use that GaugeClass provides.

Another feature I wanted was real-time updating of the gauge when the gauge is connected to another BOOPSI object. For instance, my program gauge_driver2.c connects a BOOPSI proportional gadget to a gauge object. When the user slides the knob on the proportional gadget, the gadget automatically updates the imagery in the gauge, without any intervention on the program's part! Only one initial DrawImage() call is necessary to put the gauge onto the screen. After that, Intuition does all the work involved in inter-object communication.

Since GaugeClass is a subclass of "imageclass," and therefore "rootclass", it inherits all the methods and attributes from those classes. I wrote GaugeClass so that it did not provide any new methods; it merely modifies certain methods inherited from "rootclass" and "imageclass." Any methods that GaugeClass is not interested in are handled by its superclass(es). For instance, when a GaugeClass image object receives an instruction that it should render itself, it performs a custom rendering operation. Most of the other methods, though, are handled by GaugeClass's superclasses.

GaugeClass does add some of its own attributes, however. For instance, an object needs to keep track of whether or not it has a frame, whether that frame is recessed, the color of the gauge, etc. Here is the structure that defines the data that GaugeClass keeps track of (from the file gaugeclass_private.h):

```
struct gauge_data {
struct Image *fgrect;
struct Image *bgrect;
struct Image *frame;
struct RastPort *rp;
struct DrawInfo *dri;
ULONG top, left, width, height;
ULONG fg, bg; /* pen numbers */
ULONG amount;
ULONG max;
ULONG off; /* offset */
ULONG flags;
};
```

The first three fields are all pointers to image objects. "Fgrect" and "bgrect" point to "fillrectclass" image objects. GaugeClass uses these to render the foreground and background colors of the gauge indicator rectangle. The third, optional image object is a "frameiclass" object that is used to draw a border around the gauge if the program-

mer requests one. All of these image objects are allocated when the programmer creates a new GaugeClass object, and they are all destroyed when the gauge is destroyed. These images comprise the whole gauge effect. In writing GaugeClass, I did not need to know how the "fillrectclass" and "frameiclass" images rendered themselves; I merely had to rely on the fact that they work! This illustrates another big benefit of OOP. The OOP programmer can take existing objects and combine them into a new, more specialized object, without having to duplicate a lot of code.

The next eight fields, except for "rp" and "dri," hold information that is duplicated in the three BOOPSI objects that actually comprise the gauge image, but I decided to include them here for convenience rather than request them from the objects. Keeping a copy of the "rp" and "dri" values comes in handy when doing real-time updating of the imagery, as I will explain below. The "amount" field divided by the "max" fields defines the fraction completed. At first I made this a straight "percentage of completion" image, meaning that "max" was always 100, but then I decided that the programmer might want a more precise fraction, so GaugeClass lets the programmer set both the maximum amount, and the current amount. The "off" field is a copy of the offset value passed by DrawImage(), and is also used in updating the imagery in real time. Finally, the "flags" field holds important information, like whether or not the gauge is currently drawn or erased, and in which direction the gauge "grows," left to right, right to left, top to bottom, etc. Note that the programmer does not ever need to allocate and fill in this structure; in fact the programmer is not even aware of it. GaugeClass maintains one of these structures for each gauge object that is created.

Using GaugeClass Objects in a Program

The first step in using GaugeClass is to include the files "intuition/imageclass.h", "intuition/classusr.h", and "intuition/classes.h". Finally, include "gaugeclass_public.h". An application does not need to include "gaugeclass_private.h". You also need to link "gaugeclass.o" together with your program. This file contains the GaugeClass "dispatcher" function, which handles the methods for gauge objects. If your copy of "amiga.lib" does not include the "hookEntry" function, as mine does not, you will need to assemble that and link it in as well. Finally, you need to link in the "classface.asm" routines, written by Jim Mackraz, the creator of BOOPSI.

My include file "gaugeclass_public.h" lists all the information necessary for an application to use these gauge images. All other information pertaining to GaugeClass is private, enforcing the "black-box" nature of OOP. The include file lists prototypes for the two functions necessary for initializing and deleting the class, initGaugeClass() and freeGaugeClass(). Your program should call initGaugeClass() when it is performing its initialization. This function will return a function to a class that you should hold onto until it is time for the program to terminate. At that time, pass the class pointer to freeGaugeClass() to close the class down.

Once GaugeClass is initialized, just call NewObject() with a list of initial attributes to create a gauge. For instance, the following function call will create a gauge that, when instructed to render, will appear at x-y coordinates (50,50), with a length of 100 pixels, a height of 10 pixels, and an initial percentage completed value of 45%:

```
Class *GaugeClass;
GaugeClass = initGaugeClass();
...
struct Image *MyGauge;
MyGauge = (struct Image *)NewObject(GaugeClass, NULL,
IA_Top, 50,
IA_Left, 50,
IA_Width, 100,
IA_Height, 10,
Gauge_Amount, 45,
TAG_END);
```

BOOPSI uses the Tag facilities provided by the new utility.library extensively. Tag lists, which are lists of attribute-value pairs, allow extensions of existing structures and objects without a lot of recompiling. A function is simply set up to accept a Tag list, and the attribute values in that list then serve as parameters to the function.

By default, a gauge will be created with a frame around it, and in keeping with Commodore's style guidelines, this frame will be recessed (gauges are read-only displays of information). Since these are default values, the programmer need not specify them. Also, the default foreground color is blue (pen number 3), the default background color is grey (pen number 0), and the default maximum amount is 100. The IA_Top, IA_Left, IA_Width, and IA_Height attributes are all defined by "imageclass," but since a GaugeClass object is a member of "imageclass," via inheritance, it shares these attributes. The Gauge_Amount attribute, specifying the current value of the gauge, is local to GaugeClass.

Here is a list of attributes that are defined at the GaugeClass level:

Gauge_Framed. The default value is TRUE. If this attribute is TRUE, the gauge will appear with a frame around it.

Gauge_RecessedFrame. The default value is TRUE. If the programmer wants a frame around the gauge, then this attribute indicates whether or not the frame will be recessed.

Gauge_Direction. The default value is GAUGE_DIRECTION_L2R. This attribute determines in which direction the gauge will "grow" as the amount increases with respect to the maximum amount. Acceptable values for this attribute are GAUGE_DIRECTION_L2R (left to right), GAUGE_DIRECTION_R2L (right to left), GAUGE_DIRECTION_T2B (top to bottom), and GAUGE_DIRECTION_B2T (bottom to top).

Gauge_Amount. The default value is 50. This can be any number from 0 to 232-1. It reflects the current value of the gauge.

Gauge_Max. The default value is 100. This can be any number from 1 to 232-1. It reflects the maximum amount that Gauge_Amount is allowed to be. The default value of 100 allows a quick way to construct a "percentage completed" gauge most conveniently, since this value need not be specified.

Additional "imageclass" attributes recognized are:

IA_FGPen. This attribute describes the foreground color of the gauge.

IA_BGPen. This attribute reflects the background color of the gauge.

After obtaining MyGauge, the following command will render it:

```
DrawImage (MyWindow->RPort, MyGauge, 0,0);
```

Workbench Window Icons Tools

Backdrop

Execute Command

Amazing Computing

What's the best way to improve productivity on your Workbench?

With
Amazing Computing

Amazing Computing for the Commodore Amiga, *AC's GUIDE* and *AC's TECH* provide you with the most comprehensive coverage of the Amiga. Coverage you would expect from the longest running monthly Amiga publication.

The pages of *Amazing Computing* bring you insights into the world of the Commodore Amiga. You'll find comprehensive reviews of Amiga products, complete coverage of all the major Amiga trade shows, and hints, tips, and tutorials on a variety of Amiga subjects such as desktop publishing, video, programming, and hardware. You'll also find a listing of the latest Fred Fish disks, monthly columns on using the CLI and working with ARexx, and you can keep up to date with new releases in "New Products and Other Neat Stuff."

AC's GUIDE to the Commodore Amiga is an indispensable catalog of all the hardware, software, public domain collection, services, and information available for the Amiga. This amazing book lists over 3500 products and is updated every six months!

AC's TECH for the Commodore Amiga provides the Amiga user with valuable insights into the inner workings of the Amiga. In-depth articles on programming and hardware enhancement are designed to help the user gain the knowledge he needs to get the most out of his machine.

Call 1-800-345-3360

Does this function look familiar? It is the old DrawImage() function, used to render old-fashioned images. Intuition is smart enough to distinguish old-style images from BOOPSI ones; when it encounters a BOOPSI image object, BOOPSI handles it, otherwise it is handled in the old way. The programmer is shielded from the internal details of this by the methodology of OOP. He only wants to draw the thing, he does not necessarily want to know how the object renders itself.

Once a gauge object is created, you can change an attribute at any time. For instance, to change the Gauge_Amount attribute to 60%, use the function call:

```
SetAttrs (MyGauge,
          Gauge_Amount, 60,
          TAG_END);
```

You can set more than one attribute at a time, too, simply by adding the attribute label and value in the function call before the TAG_END. Note that setting some attributes has a different effect than setting others, and some attributes are not "settable" at all once the object is created. For instance, you can change the IA_Top attribute at any time, but you will not notice a change until you instruct the object to render again with a call to DrawImage(). However, changing the Gauge_Amount or Gauge_Max attributes will change the imagery immediately, and no subsequent call to DrawImage() is necessary. Gauge_Amount and Gauge_Max are the only GaugeClass attributes that are settable; Gauge_Direction, Gauge_Framed, and Gauge_RecessedFrame are not settable once the gauge is created. All of the IA attributes are settable.

You can also get attribute values, one at a time, from an object. This is done via a call to GetAttrs(), as in the following example:

```
ULONG value;
BOOL success;
success = GetAttrs(Gauge_Amount, MyGauge, &value);
```

If the attribute is "gettable" (check variable "success" after the function returns) then the attribute's value will be placed into the variable "value." Not all attributes are gettable. In the case of GaugeClass, all the IA attributes are gettable, but only Gauge_Amount and Gauge_Max are gettable. I did this just as an exercise. In principle all of GaugeClass's attributes could be gettable.

That is basically it as far as using GaugeClass in your program. Just remember to call DisposeObject() on your gauge image when your program is through with it, and to free GaugeClass:

```
DisposeObject (MyGauge);
freeGaugeClass (GaugeClass);
```

To get an idea of how to use these objects, look at my driver programs, gauge_driver1.c and gauge_driver2.c.

But How Does GaugeClass Work?

Any class operates by implementing what is referred to in OOP circles as a "dispatcher." I mentioned a few times above that an object can be instructed to perform some action, or method. It is the dispatcher's responsibility to sort through the methods that its objects receive and perform the appropriate action. Since the class "knows" the real internal structure of its object, it knows precisely what functions to call on that object's data. On the other hand, some methods that an object receives may not be known to the object's class at all. For instance, one of the methods defined at the "rootclass" level is a method that asks the receiving object to add itself to a list. GaugeClass simply hands off that message to its superclass, since it does not understand it. The "imageclass" class does not understand it either, so eventually the message works its way up to "rootclass," which handles the method in the appropriate way. Again, the actual steps needed to perform this operation are irrelevant, as far as GaugeClass and "imageclass" are concerned. They just need to trust that "rootclass" will do its job.

Another possibility is that a particular class will redefine an existing method. For instance, the method to dispose of an object is defined at the "rootclass" level. So what does GaugeClass do when one of its objects receives a message to dispose of itself? First, the dispatcher has to dispose of the image objects it created to make up that object's particular imagery, and then it passes the message to its superclass, "imageclass." Again, the message eventually works its way up to "rootclass," which disposes of the whole object. That's all there is to it! GaugeClass does not know what goes on behind the scenes as far as BOOPSI goes, and does not care. It is only responsible for its objects, and simply allows the superclasses to take care of the rest.

The dispatcher handles setting and getting attributes similarly. If the attribute is local to, or redefined by, the dispatcher's class, the dispatcher just handles the message. Otherwise, it could be a message to set/get an attribute belong to a superclass, so the message is forwarded up the chain. Any return value is eventually returned via this dispatcher to the application.

An object can also send itself messages, and this is how GaugeClass handles updating its imagery in real time. When the GaugeClass dispatcher receives a message to set Gauge_Amount or Gauge_Max, it sets these values, recalculates the imagery dimensions, and sends a "render" message to itself. Another example of this is if something goes wrong during the creation phase of a new gauge. The gauge simply tells itself to dispose of itself.

DoMethod() et al

The Intuition functions NewObject(), DisposeObject(), SetAttrs(), and GetAttrs() are actually just interfaces to the real workhorses, DoMethod() and its relatives in classface.asm. These functions format the messages in the correct way for the object's dispatcher function to understand them. I recommend that you take a look at the source code for DoMethod() and the others.

Sending a message to an object means sending the object a method with a list of parameters required by that method. Some methods, such as the "dispose" method, do not require any parameters; others, such as the "new object" method, require a bunch. DoMethod() packages them up and sends them to the object. DoMethod is called in the following way:

```
returnvalue = DoMethod(object_pointer, method_id, parameter1,
...);
```

The return_value variable contains any return value returned by that particular method. Some methods return some value, others do not.

The "rootclass" class defines ten methods. Some of them are redefined by GaugeClass, but for the most part, GaugeClass lets "rootclass" handle them. The "rootclass" methods are

OM_NEW. This method creates a new object. Applications should call the Intuition function NewObject(), which is a "wrapper" for this method.

OM_DISPOSE. This method tells an object to dispose of itself. GaugeClass has to deal with this method to some extent, since it has to dispose of the BOOPSI objects it created for the object. Applications should call the Intuition function DisposeObject() to apply this method to an object.

OM_GET. This method gets an attribute value, if possible, from an object. GaugeClass handles this method, too. If it does not understand what the requested attribute is, it passes the method up to its superclass, since the attribute could belong there. Applications can use the Intuition function GetAttrs() instead of calling this method directly via DoMethod().

OM_SET. This method sets an object's attribute(s), if possible. GaugeClass handles this message like it handles OM_GET. If it understands the attribute and can update it, it does. Otherwise, the superclass gets a crack at it. Applications should call SetAttrs() for this method.

OM_UPDATE. This message is only sent from one BOOPSI object to another. It informs the target object that it should update some attribute value. This is how, in my second driver program, the proportional gadget updates the gauge. It sends its STRINGA_LongVal attribute to the gauge object in an OM_UPDATE message. Since GaugeClass does not understand what STRINGA_LongVal means, the OM_UPDATE method "maps" STRINGA_LongVal to Gauge_Amount. This mapping is part of the proportional gadget's attributes. GaugeClass handles this method exactly as if it had received an OM_SET message.

OM_ADDTAIL, OM_REMOVE. These methods instruct an object to add itself to an Exec list, and to remove itself from an Exec list, respectively. GaugeClass sends this method to its superclass, since it is not interested in redefining it.

OM_ADDMEMBER, OM_REMMEMBER. These methods are for those objects that maintain internal lists of objects. Since GaugeClass does not, it simply passes this method to its superclass for it to handle.

OM_NOTIFY. This method tells on object to send OM_UPDATE message to any objects on its internal "broadcast" list. Since GaugeClass does not broadcast its values to anything, this method is ignored (passed to the superclass).

The "imageclass" class, in turn, defines several methods particular to images. The only ones recognized by GaugeClass are the following:

IM_DRAW, IM_ERASE. These methods instruct an object to draw and erase itself, respectively. Applications call these methods by using the Intuition functions DrawImage() and EraseImage(). GaugeClass does some custom rendering, so it modifies the meaning of these methods.

The other "imageclass" methods, **IM_HITTEST, IM_DRAWFRAME, IM_HITFRAME, IM_ERASEFRAME,** and **IM_FRAMEBOX** are ignored by GaugeClass. I chose not to implement these methods in order to make GaugeClass as simple as possible.

Finally

Examine the source for gaugeclass.c. I think that it is simple enough that you should be able to follow it without much explanation. Note that gaugeclass.c uses two custom include files, "gaugeclass_private.h" and "gaugeclass_public.h". As the name implies, "gaugeclass_private.h" is private information to GaugeClass, and no application besides GaugeClass should need to include it.

There are many ways to improve GaugeClass. One way would be to add text to the gauge. Another would be to add tick marks along the side. I decided not to implement these features, because I wanted to keep the class simple. As I said earlier, simplicity comes at the cost of flexibility. For instance, although GaugeClass uses two "fillrectclass" objects to render its gauge indicator bar, it does not allow the full capability of these objects, such as pattern fills. One way to add this capability would be to define two more attributes for GaugeClass, say Gauge_FGRect and Gauge_BGRect. These attributes would describe "fillrectclass" objects that the user would have to create himself. This would not require much modification to the source code, but I wanted the user to create only one object, and not have to maintain two others himself.

Feel free to modify the source code and experiment in any way you want. Let me know if you improve the class significantly; after all, I am still learning a lot about this great part of the Amiga's operating system myself!

GAUGECLASS_PUBLIC.H

```
/*
** GAUGECLASS_PUBLIC.H
** PUBLIC INFORMATION ABOUT GAUGECLASS
**
** This is all an application needs to know about our
** custom class.  The details of the class's operation
** are secret!
*/


/* ATTRIBUTES */
#define Gauge_offset          0x10000
#define Gauge_Dummy           (TAG_USER + Gauge_offset)
#define Gauge_Framed          (Gauge_Dummy + 0x01)
#define Gauge_RecessedFrame   (Gauge_Dummy + 0x02)
#define Gauge_Direction       (Gauge_Dummy + 0x03)
#define Gauge_Amount          (Gauge_Dummy + 0x04)
#define Gauge_Max             (Gauge_Dummy + 0x05)


/* DEFINED ATTRIBUTE VALUES */
/* These are for Gauge_Direction: */
#define GAUGE_DIRECTION_L2R   0x00000000
#define GAUGE_DIRECTION_R2L   0x00000100
#define GAUGE_DIRECTION_T2B   0x00000200
#define GAUGE_DIRECTION_B2T   0x00000300


/* FUNCTIONS CALLABLE BY THE APPLICATION */
extern Class *initGaugeClass();
extern BOOL freeGaugeClass(Class *cl);
```

gauge_driver1.c

```
/* gauge_driver1.c */


#include <exec/types.h>
#include <intuition/intuition.h>
#include <intuition/imageclass.h>
#include <intuition/classusr.h>
#include <intuition/classes.h>
#include <dos/dos.h>
#include <proto/exec.h>
#include <proto/intuition.h>

#include "gaugeclass_public.h"


#define MINLIB 37

#define GREY    0
#define BLACK   1
#define WHITE   2
#define BLUE    3

#define WINDOWLEFT     0
#define WINDOWTOP      0
#define MINWINWIDTH    320
#define MINWINHEIGHT   200
#define MAXWINWIDTH    640
#define MAXWINHEIGHT   400
#define MY_IDCMP       (IDCMP_CLOSEWINDOW)
#define MY_FLAGS       (WFLG_DEPTHGADGET | \
                        WFLG_SIZEGADGET  | \
                        WFLG_DRAGBAR     | \
                        WFLG_CLOSEGADGET | \
                        WFLG_SMART_REFRESH)

#define GAUGE_TOP      30
#define GAUGE_LEFT     30
#define GAUGE_WIDTH    100
#define GAUGE_HEIGHT   15
#define INITIALVAL     75


UBYTE                *vers="\0$VER: gauge_driver1";
char                 *windowtitle="GaugeClass Driver
#1";
struct IntuitionBase  *IntuitionBase = NULL;
struct Window          *MyWindow     = NULL;
struct DrawInfo        *MyDrawInfo   = NULL;
Class                 *GaugeClass    = NULL;
struct Image           *Gauge        = NULL;

void
open_all(),           /* open everything  */
close_all(),          /* close everything */
abort(char *, SHORT);   /* abort            */

/* disable SAS/C CTRL-C trapping... */
int CXBRK(void)     { return (0); }
int chkabort(void)  { return (0); }


/*------------------------------*/
void
main()
{  struct IntuiMessage *imsg;
    ULONG              msgclass, idcmp_signal, signals;
open_all();
idcmp_signal = 1L << MyWindow->UserPort->mp_SigBit;
FOREVER {
  signals = Wait(idcmp_signal | SIGBREAKF_CTRL_C);
   if (signals & idcmp_signal)  {
      while (imsg =
       (struct IntuiMessage *)GetMsg(MyWindow-
>UserPort)) {
        msgclass = imsg->Class;
       ReplyMsg ((struct Message *)imsg);
        switch (msgclass) {
          case IDCMP_CLOSEWINDOW:
              abort (NULL, 0);
               break;
            }
         }
      }
  if (signals & SIGBREAKF_CTRL_C)  abort (NULL, 0);
   }
}
/*------------------------------*/



void
open_all()
{
IntuitionBase = (struct IntuitionBase *)
  OpenLibrary("intuition.library", MINLIB);
if (!IntuitionBase)
   abort("Error opening intuition library!", 5);
MyWindow = OpenWindowTags(NULL,
   WA_Left,      WINDOWLEFT,
   WA_Top,       WINDOWTOP,
   WA_Width,     MINWINWIDTH,
   WA_Height,    MINWINHEIGHT,
  WA_DetailPen, GREY,
   WA_BlockPen,  BLACK,
   WA_IDCMP,     MY_IDCMP,
   WA_Flags,     MY_FLAGS,
  WA_MinWidth,  MINWINWIDTH,
  WA_MinHeight, MINWINHEIGHT,
  WA_MaxWidth,  MAXWINWIDTH,
  WA_MaxHeight, MAXWINHEIGHT,
   WA_Title,     windowtitle,
   TAG_END);
if (!MyWindow)
   abort ("Error opening window!", 5);
MyDrawInfo = GetScreenDrawInfo(MyWindow->WScreen);
if (!MyDrawInfo)
   abort ("Error getting drawinfo!", 5);
GaugeClass = initGaugeClass();
if (!GaugeClass)
   abort ("Error initializing gaugeclass!", 5);
Gauge = (struct Image *)NewObject(GaugeClass, NULL,
   IA_Top,          GAUGE_TOP,
   IA_Left,         GAUGE_LEFT,
   IA_Width,        GAUGE_WIDTH,
   IA_Height,       GAUGE_HEIGHT,
   Gauge_Amount,    INITIALVAL,
  Gauge_Direction, GAUGE_DIRECTION_L2R,
   TAG_END);
if (!Gauge)
   abort ("Error allocating gauge object!\n", 5);
```

```
DrawImage (MyWindow->RPort, Gauge, 0,0);
Delay (60);
SetAttrs (Gauge, Gauge_Amount, 30, TAG_END);
Delay (60);
SetAttrs (Gauge, Gauge_Max, 300, TAG_END);
}



void
close_all()
{
if (Gauge) DisposeObject (Gauge);
if (GaugeClass) freeGaugeClass(GaugeClass);
if (MyWindow) {
 FreeScreenDrawInfo (MyWindow->WScreen, MyDrawInfo);
 CloseWindow(MyWindow);
  }
if (IntuitionBase) CloseLibrary (IntuitionBase);
}



void
abort(char *txt, SHORT errorcode)
{
close_all();
if (txt) puts(txt);
exit(errorcode);
}
```

gauge_driver2.c

```
/* gauge_driver2.c */

#include <exec/types.h>
#include <intuition/intuition.h>
#include <intuition/imageclass.h>
#include <intuition/classusr.h>
#include <intuition/classes.h>
#include <intuition/gadgetclass.h>
#include <intuition/icclass.h>
#include <dos/dos.h>
#include <proto/exec.h>
#include <proto/intuition.h>

#include "gaugeclass_public.h"


#define MINLIB 37

#define GREY   0
#define BLACK  1
#define WHITE  2
#define BLUE   3

#define WINDOWLEFT     0
#define WINDOWTOP      0
#define MINWINWIDTH    320
#define MINWINHEIGHT   200
#define MAXWINWIDTH    640
#define MAXWINHEIGHT   400
#define MY_IDCMP       (IDCMP_CLOSEWINDOW)
#define MY_FLAGS       (WFLG_DEPTHGADGET | \
                        WFLG_SIZEGADGET  | \
                        WFLG_DRAGBAR     | \
                        WFLG_CLOSEGADGET | \
                        WFLG_SMART_REFRESH)

#define GAUGE_TOP      30
#define GAUGE_LEFT     30
#define GAUGE_WIDTH    100
#define GAUGE_HEIGHT   15

#define INITIALVAL     75

#define PROPGADWIDTH   10
#define PROPGADHEIGHT  80
#define VISIBLE        10
#define TOTAL          110
#define PGA_NewLook    (PGA_Dummy + 0x000A)

UBYTE                 *vers="\0$VER: gauge_driver2";
char                  *windowtitle="GaugeClass Driver
#2";
struct IntuitionBase  *IntuitionBase = NULL;
struct Window          *MyWindow     = NULL;
struct DrawInfo        *MyDrawInfo   = NULL;
Class                 *GaugeClass    = NULL;
struct Image           *Gauge        = NULL;
struct Gadget          *Propgad      = NULL;

struct TagItem prop2gaugemap[] = {
  {PGA_Top, Gauge_Amount}, {TAG_END}
   };

void
open_all(),           /* open everything  */
```

```
close_all(),          /* close everything */
abort(char *, SHORT);   /* abort          */

/* disable SAS/C CTRL-C trapping... */
int CXBRK(void)     { return (0); }
int chkabort(void)  { return (0); }


/* ------------------------------ */
void
main()
{  struct IntuiMessage *imsg;
    ULONG             msgclass, idcmp_signal, signals;
open_all();
idcmp_signal = 1L << MyWindow->UserPort->mp_SigBit;
FOREVER {
  signals = Wait(idcmp_signal | SIGBREAKF_CTRL_C);
   if (signals & idcmp_signal)  {
      while (imsg =
       (struct IntuiMessage *)GetMsg(MyWindow-
>UserPort)) {
        msgclass = imsg->Class;
       ReplyMsg ((struct Message *)imsg);
        switch (msgclass) {
          case IDCMP_CLOSEWINDOW:
              abort (NULL, 0);
               break;
            }
         }
      }
  if (signals & SIGBREAKF_CTRL_C)  abort (NULL, 0);
   }
}
/* ------------------------------ */



void
open_all()
{
IntuitionBase = (struct IntuitionBase *)
  OpenLibrary("intuition.library", MINLIB);
if (!IntuitionBase)
   abort("Error opening intuition library!", 5);
MyWindow = OpenWindowTags(NULL,
   WA_Left,      WINDOWTOP,
   WA_Top,       WINDOWLEFT,
   WA_Width,     MINWINWIDTH,
   WA_Height,    MINWINHEIGHT,
  WA_DetailPen, GREY,
  WA_BlockPen,  BLACK,
   WA_IDCMP,     MY_IDCMP,
   WA_Flags,     MY_FLAGS,
  WA_MinWidth,  MINWINWIDTH,
  WA_MinHeight, MINWINHEIGHT,
  WA_MaxWidth,  MAXWINWIDTH,
  WA_MaxHeight, MAXWINHEIGHT,
   WA_Title,     windowtitle,
   TAG_END);
if (!MyWindow)
   abort ("Error opening window!", 5);
MyDrawInfo = GetScreenDrawInfo(MyWindow->WScreen);
if (!MyDrawInfo)
   abort ("Error getting drawinfo!", 5);
GaugeClass = initGaugeClass();
if (!GaugeClass)
   abort ("Error initializing gaugeclass!", 5);
Gauge = (struct Image *)NewObject(GaugeClass, NULL,
   IA_Top,             GAUGE_TOP,
   IA_Left,            GAUGE_LEFT,
   IA_Width,           GAUGE_WIDTH,
   IA_Height,          GAUGE_HEIGHT,
   Gauge_Amount,       INITIALVAL,
  Gauge_Direction,    GAUGE_DIRECTION_L2R,
   TAG_END);
if (!Gauge)
   abort ("Error allocating gauge object!", 5);
DrawImage (MyWindow->RPort, Gauge, 0,0);
Propgad = (struct Gadget *)NewObject(NULL, "propgclass",
   GA_ID, 1,
  GA_Top, MyWindow->BorderTop + 5,
  GA_Left, MyWindow->BorderLeft + 5,
  GA_Width, PROPGADWIDTH,
  GA_Height, PROPGADHEIGHT,
   GA_RelVerify, TRUE,
   PGA_Total, TOTAL,
   PGA_Top, INITIALVAL,
   PGA_Visible, VISIBLE,
   PGA_NewLook, TRUE,
   ICA_TARGET, Gauge,
  ICA_MAP, prop2gaugemap,
   TAG_END);
if (!Propgad)
   abort ("Error allocating proportional gadget!", 5);
AddGList (MyWindow, Propgad, -1,-1, NULL);
RefreshGList (Propgad, MyWindow, NULL, -1);
}



void
close_all()
{
if (Propgad) DisposeObject(Propgad);
if (Gauge) DisposeObject (Gauge);
if (GaugeClass) freeGaugeClass(GaugeClass);
if (MyWindow) {
 FreeScreenDrawInfo (MyWindow->WScreen, MyDrawInfo);
 CloseWindow(MyWindow);
  }
if (IntuitionBase) CloseLibrary (IntuitionBase);
}



void abort(char *txt, SHORT errorcode)
{
close_all();
if (txt) puts(txt);
exit(errorcode);
}
```

✔

The complete set of listings can be found on the AC's TECH disk.

Please write to:
Scott Palmateer
c/o AC's TECH
P.O. Box 2140
Fall River, MA 02722

Comeau C++ 3.0.1 With Templates Available. Affordable. Reliable. Supported.

Widely Available
Not only has Comeau C++ been ported to multiple operating systems and platforms, it is uncontestably the widest available commercial C++ anywhere. AmigaDOS, UNIX, MS-DOS, SunOS, RS/6000, SCO, SVR3, SVR4, XENIX and then some. You name it, we're there. And all with the same robust and up-to-date functionality.

Standard 3.0 ARM
Other C++'s claim conformance to AT&T USL's cfront, some still at the outdated cfront 2.0 or cfront 2.1 level. Comeau C++ was written as per the ARM and literally licensed from AT&T USL, so we don't need to claim anything. We literally are cfront. Further, we're 3.0.1, the latest release.

Latest Features
Templates, full nested types, multiple inheritance. They're all here. We are active and voting members of the ANSI C++ committee, so the latest working draft document changes will always be available as their specifications mature and become appropriate.

Excellent Reviews
Comeau C++ continues to get international acclaim in magazines such as *The C++ Journal*, *Dr. Dobb's Journal*, *Computer Language* and others. On the Amiga, recent coverage includes *AC's Tech* as well as *AmigaWorld* magazine. "netters" in comp.sys.amiga.reviews love it too.

Supports 3rd Party Libraries
We work with many of the 3rd party library vendors to ensure their libraries are consistent, reliable and language conforming. This means that most general purpose, niche specific, commercial and public domain libraries work unchanged with Comeau C++.

Amiga Support
Comeau C++ has been available on the Amiga for over two years. It is seriously supported and always kept up-to-date. Often it's even the first to get a feature. Comeau C++ is a translating compiler that generates C as its object code. On the Amiga, back end C compiler support includes SAS/C and Manx C. Dice C support is currently in beta. GCC support is currently in the works. Contact us for the latest news.

International Leadership
You don't need to wait weeks for your order. Comeau C++ is shipped internationally via UPS next day air. Add in a day or two for customs and it is there! Throw in a plethora of electronic mechanisms to contact us, and you might as well be next door.

Comeau Components
Imagine access to sets of class libraries that have had actual extensive use at the production level by the originators of C++ (AT&T, USL, etc.) since 1987. These reliable classes are what the Comeau Components are about. All designed and proven in the real world to have maximum run-time efficiency, reuse, productivity and functionality, allowing you to concentrate on your application's specifics. AmigaDOS specific additions will be available soon. Contact us for the latest news.

We Stand By What We Say
Far too many companies are market and sales driven. Comeau Computing is consumer driven. We'll stand by our product. Whether our free domestic shipping, much touted free technical support, 60-day offer or often free upgrades, we'll break the barriers. We're consumers too, and know you deserve better!

Easy Access
You'll always have fast and easy access to our sales, service and support staff. We can be conveniently accessed via many mechanisms, some accessible 24 hours/7 days a week, including: mail (91-34 120th Street, Richmond Hill, NY, 11418-3214), UseNet (acg@csanta.attmail.com), BIX (comeau), Compuserve (72331,3421), Prodigy (tshp50a), Voice (718-945-0009), Fax (718-441-2310).

Comeau Computing
91-34 120th Street
Richmond Hill, NY 11418

Comeau C++.... Anything less is A-

List of Advertisers

Company	Page Number
AMOS	CII
Armadillo Computing	15
Comeau Computing	39
Delphi Noetic Systems	37
Devine Computers	11
Digital Creations	CIV
Digital Imagery	7
Memory Management	17
Oregon Research	5
Oxxi	CIII

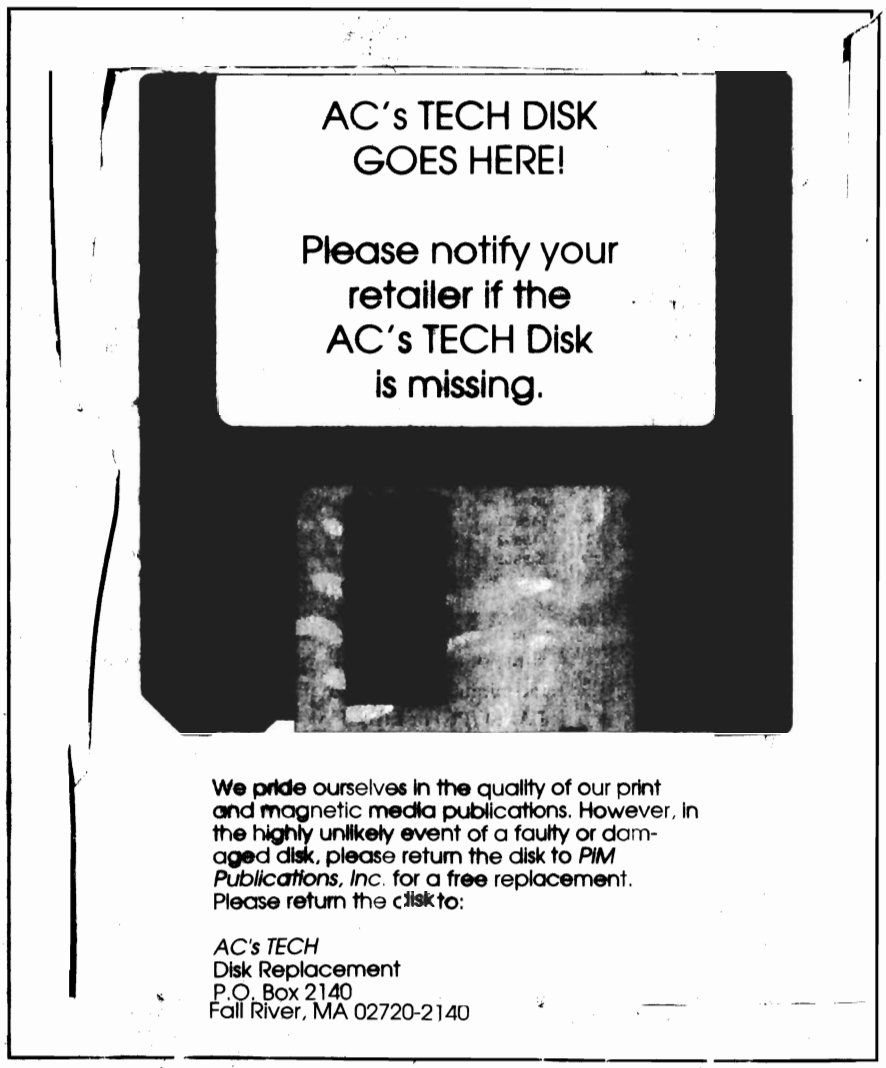
AC's TECH Disk
Volume 4, Number 1

A few notes before you dive into the disk!

- You need a working knowledge of the AmigaDOS CLI as most of the files on the AC's TECH disk are only accessible from the CLI.

- In order to fit as much information as possible on the AC's TECH Disk, we archived many of the files, using the freely redistributable archive utility 'lharc' which is provided in the C: directory. lharc archive files have the filename extension .lzh.

  To unarchive a file *foo.lzh*, type *lharc x foo*
  For help with lharc, type *lharc ?*
  Also, files with 'lock' icons can be unarchived from the WorkBench by double-clicking the icon, and supplying a path.



We pride ourselves in the quality of our print and magnetic media publications. However, in the highly unlikely event of a faulty or damaged disk, please return the disk to *PiM Publications, Inc.* for a free replacement. Please return the disk to:

AC's TECH
Disk Replacement
P.O. Box 2140
Fall River, MA 02720-2140

Be Sure to Make a Backup!

CAUTION!

Due to the technical and experimental nature of some of the programs on the AC's TECH Disk, we advise the reader to use caution, especially when using experimental programs that initiate low-level disk access. The entire liability of the quality and performance of the software on the AC's TECH Disk is assumed by the purchaser. PiM Publications, Inc, their distributors, or their retailers, will not be liable for any direct, indirect, or consequential damages resulting from the use or misuse of the software on the AC's TECH Disk. (This agreement may not apply in all geographical areas.)

Although many of the individual files and directories on the AC's TECH Disk are freely redistributable, the AC's TECH Disk itself and the collection of individual files and directories on the AC's TECH Disk are copyright PiM Publications, Inc, and may not be duplicated in any way. The purchaser, however, is encouraged to make an archive/backup copy of the AC's TECH Disk.

Also, be extremely careful when working with hardware projects. Check your work twice, to avoid any damage that can happen. Also, be aware that using these projects may void the warranties of your computer equipment. PiM Publications, or any of its agents, is not responsible for any damages incurred while attempting this project.

Programming the Amiga in Assembly Language

There are usually several ways to improve the performance of your source code when writing an assembly language program. The most obvious is to consider a rewrite of the mechanics of the program. For example, there are several ways to write a Mandelbrot program. The simplest sets every point to some color while the more complicated ones outline the set and then automatically fill in the color. The code can be very long but will result in a much quicker program. This type of rewriting, however, is very individualized and will be unique for each program.

Relative Code

There are some general procedures you can consider, though, with any program. First, try to write your code with all labels or variables as "position independent." That is, your code does not refer to the variable location but rather its distance from where you are in the program at that point—much like a branch would do. For example, instead of putting the contents of LABEL2 into register d0, put the information, say, 300 bytes ahead into d0. This is done by referencing the variable as LABEL(PC); the "pc" means position counter and will cause the assembler to compute a distance rather than location. But, what does this do to our code? Let's write a short program and look at the assembled code.

```
START   MOVE.L   ACROSS,D0      2039 0000 000C
        MOVE.L   DOWN(PC),D1    223A 0008
        RTS
ACROSS  DC.L     0
DOWN    DC.L     0
        END
```

Notice the difference in Line 1 and Line 2. With just a small change in the source code, we saved two bytes; while this isn't very much, if this code was within an iteration loop of 1000 times, that would be a savings of 2000 bytes and an increase in speed.

Unfortunately, only the source operand can be written in PC format, not the destination operand; you can't write MOVE.L #1,DATA1(PC), at least not with the current crop of assemblers. But there is a technique that allows you to reference any variable as a distance from a fixed variable whose address is stored in a register, usually a4 or a5. This will add one more line of code when you put the initial variable address in the register. Let's store the start of all our variables, call it V, in register a5. Now any location after V can be referred to as LABEL-V(A5); again, this computes a distance not an address. Now look at a variation of our first program.

```
START   LEA      V(PC),A5               4BFA 002C
        MOVE.L   ACROSS(PC),D0          203A 001C
        MOVE.L   DOWN-V(A5),D1          222D 0004
        MOVE.L   #1,LABEL1              23FC 0000 0001 0000 002A
        MOVE.L   #2,LABEL2-V(A5)        2BFC 0000 0002 000C
        MOVEQ    #3,D2                  7403
        RTS
V:
ACROSS  DC.L     0
DOWN    DC.L     0
LABEL1  DC.L     0
LABEL2  DC.L     0
LABEL3  DC.L     0
        END
```

The first line stores the address of our variable storage area in a5. Be sure not to change a5 during the loop in which you're using this technique. The code length for the next two lines is the same so using LABEL(PC) or LABEL-V(A5) creates identical length code for the source operand. The next two lines show how the code length can be decreased using the destination operand as a register offset. By the way, the address of V is the same as the address of ACROSS; I find it easier to add V at the beginning and any macro using this technique won't have to be rewritten.

Notice that the last line shows how MOVEQ will create shorter code that MOVE.L for immediate values between -128 and +127. In fact, MOVEQ #0 is even quicker than the code CLR. And it's quicker to double or halve a number with a LSL or LSR rather than use MUL or DIV.

Complex Functions

by William P. Nee

The quickest way to use variables is to assign them to registers at the start of the program. Since this does limit the register's use, you probably want to do this with only two or three of your most used variables such as a counter or across/down. You can still use those registers outside a loop but leave them alone during any loop when they are being used as their assigned register. Now let's see how our code could look.

```
ACROSS    EQUR      D4
START     LEA       V(PC),A5                    4BFA 0016
          MOVE.L    DOWN(PC),D0                 203A 0012
          MOVE.L    ACROSS,D1                   2204
          MOVEQ     #3,D0                       7003
          MOVE.L    D0,LABEL1-V(A5)             2B40 0004
          MOVE.L    LABEL2-V(A5),LABEL3-V(A5)   2B6D 0008 000C
          RTS
V:
DOWN      DC.L      0
LABEL1    DC.L      0
LABEL2    DC.L      0
LABEL3    DC.L      0
          END
```

Since ACROSS was equated to register d4 at the start of the program, we don't reserve space for it at the end. You could, and probably should, also equate DOWN to a register; I didn't just to show you the difference in code length. You can also see that it's quicker to move an immediate number into a register and then move that register into a label instead of moving the number directly into the label.

To demonstrate some of these techniques I wrote a program, using an Amiga 3000, that iterates the complex functions SIN(Z), COS(Z), SINH(Z), COSH(Z), and EXP(Z). Points are set when the real or imaginary parts of Z exceed specific values. More on this later.

First, I rewrote the DPMATHMACROS.I code or, more specifically, the MOVEDP portion of it. Now if a label is going to be moved to a register, the macro uses LABEL(PC). Since most DP locations use two consecutive registers and label locations, the complete portion of the macro would read

```
          MOVE.L    LABEL1(PC),D0
          MOVE.L    LABEL1+4(PC),D1
```

This will move both long word parts of LABEL1 into d0/d1. How would you write this if the destination operand is a label? Simple; use the V offset twice. For example, to move the contents of d0/d1 to LABEL2 use code such as

```
          MOVE.L    D0,LABEL2-V(A5)
          MOVE.L    D1,LABEL2+4-V(A5)
```

This is the equivalent of MOVEDP D0,LABEL2. If you make this a macro, you must always use the V offset from a5. I find it easier to incorporate these two lines into my source code instead. This leaves me free to decide which register to use or even not to use this technique.

Complex Functions

As I mentioned earlier, this program will iterate functions of Z, or more precisely, (C1+iC2)*FN(Z)+(F1+iF2). This will let you multiply the function by a complex number, add a complex number, or both. Later on I'll give you some specific values to try with different functions and suggest a way to make an animation out of one of these functions.

Since a lot of my programs use complex numbers (numbers which include i as part of their value) I made an include file for them called COMPLEX.I. It's Listing 1 in the article and on the magazine disk. Most of the complex functions we'll use in this article involve the hyperbolic function (EXP(B)+EXP(-B))/2. For example, the sine of Z (where Z=a+ib) is:

```
SIN(a)*(EXP(b)+EXP(-b))/2+i*COS(a)*(EXP(b)-EXP(-b))/2
```

If you multiply this by (c1+ic2) and add (f1+if2) you'll get:

```
REAL=c1*SIN(a)*(EXP(b)+EXP(-b))/2-c2*COS(a)*(EXP(b)-EXP(-b))/2+f1
IMAG=c2*SIN(a)*(EXP(b)+EXP(-b))/2+c1*COS(a)*(EXP(b)-EXP(-b))/2+f2
```

To save a lot of repetition, I first compute an e1 and e2 where e1=SIN(a)*(EXP(b)+EXP(-b))/2 and e2=COS(a)*(EXP(b)-EXP(-b))/2. Now SIN(Z) reduces to:

```
REAL SIN(Z)=C1*E1-C2*E2+F1
IMAG SIN(Z)=C2*E1+C1*E2+F2
```

Substitute these new real and imaginary values for a and b respectively. Finally, put the value of b back in registers d0/d1; you'll see why in a minute.

Each function is iterated 31 times. If a specific part of the function we're using exceeds a given value, then set that point to a color corresponding to the iteration count. The part of the function you pick and the value you compare this to make up, to a great extent, the display. In these five examples all the values are compared to 50 but you can easily change that in the source code. The parts of Z compared to 50 are

```
SIN(Z) - ABS(B)          COS(Z) - ABS(B)
SINH(Z) - ABS(A)         COSH(Z) - ABS(A)
EXP(Z) - (A)
```

COS(Z) is computed much as SIN(Z). Its original value is:

```
COS(a)*(EXP(b)+EXP(-b))/2-i*SIN(a)*(EXP(b)-EXP(-b))/2
```

The hyperbolic functions are the opposite of the previous two functions. Swap a and b and change the exponential signs. So, SINH(Z) is

```
COS(b)*(EXP(a)-EXP(-a))/2+i*SIN(b)*(EXP(a)+EXP(-a))/2
```

And COSH(Z) is

```
COS(b)*(EXP(a)+EXP(-a))/2+i*SIN(b)*(EXP(a)-EXP(-a))/2
```

Of course, multiply each function by (c1+ic2) and add (f1+if2). That leaves the exponent function EXP(Z). This can be written as EXP(a)*EXP(ib). Now EXP(ib) is also COS(b)+i*SIN(b) so the entire function can be written as EXP(a)*COS(b)+i*EXP(a)*SIN(b). Let e1=EXP(a)*COS(b) and e2=EXP(a)*SIN(b) and the parts of exponent Z are

```
REAL EXP(Z)=C1*E1-C2*E2+F1
IMAG EXP(Z)=C2*E1+C1*E2+F2
```

For this function compare the value of the real part of (a) with 50 to see if a point will be set. This function should be iterated more than 31 times—more like 150 times—but I wanted to keep the program simple. You could add another menu selection for specific iteration counts.

Notice that this include file uses (pc) code whenever possible and assumes that variable location V will be stored in register a5; all the variables (e1, e2, etc.) must be located within the parent program.

The Listing

Now let's look at the entire program, Listing 2. After the include files, some variables are equated to registers, followed by three macros; notice that there are two different message check macros since I didn't want any menu check while the actual computations are being carried out. The various libraries are opened (be sure to have both the MATHIEEEDOUBBAS and MATHIEEEDOUBTRANS libraries in your LIBS:). A short cover screen reminds you of what the program hotkey options are. You could incorporate all these keys into menu selections instead. Bypass this screen by removing the REM in front of "; JMP SETUP".

After opening the screen and window, eight gadgets will appear for C1, iC2, F1, iF2, XLEFT, XRIGHT, YBOT, and YTOP. The first two are the complex number to multiply the function by; next is the complex number to add; the last four are the range of values to be shown on the screen. Default values appear initially within these gadgets, but you can change them in the source code to any values you want. The first message check waits to see what menu item (function) you pick. The macro eval_menunumber puts the item number (0-4) in TYPE; then the menu is removed.

Next the value in each gadget is converted to a DP number and stored in its correct location; each location is a double longword. A requester will ask if you want the small (128X128) or large (320X200) version and scale the coordinates accordingly. The routine SHOWIT starts the actual computations and drawing. I left some of the MOVEDP lines in with a REM to remind you how they're actually being written relative to V(a5). Notice that you can't use MOVEQ #0 for DOWN, ACROSS, and COUNT since they're equated to address registers.

Depending on your menu item selection stored in TYPE, the desired function macro is called and depending on the function, part of Z is compared to 50. If it's lower, the program continues, but if it's higher, that point is set according to the iteration count value. After each point is set the ACROSS distance is increased by XINC. The message check this time waits to see if you change palettes (#0-#9), want to quit (q), change picture size (x), return to the coordinate menu (c), or zoom by pressing the LMB. When you've finished with the zoom box, a requester will appear asking if it's O.K. to zoom. If so, a second requester will confirm the size. The new coordinates will be rescaled and the new picture drawn.

Options

These same routines are possible when the picture is completed. You can change palettes, go back to the coordinate menu, quit the program, or zoom. The end of the listing closes the menu, window, screen, and libraries. Notice that the variables after MYWINDOW start with V. In cases where a variable is used only once (zoom, palette, etc.), I do not use the V offset; that's reserved for loops where the savings in time is more noticeable.

There are 10 palettes that can be picked at any time. Some look better with different functions and at different zoom levels. The GADGETBUFFER contains the default value for each of the eight gadgets. Feel free to change these to any other values. Some suggested parameters for trying the functions are

	C1	iC2	F1	iF2	XLEFTX	RIGHT	YBOT	YTOP
SIN(Z)	1	0	0	0	-4	4	-4.5	-4.5
	1	.109	0	0	-3	3	-3	3
COS(Z)	2.95	0	0	0	-1	1	-1	1
COSH(Z)	1	0	-2.5	0	-2.25	2.25	-3	3
EXP(Z)	.4	0	0	0	-1	4	-2.5	2.5
	4	0	0	0	-.028	2.35	-1.28	1.28

You can make a very interesting animation using SIN(Z). Keep C1 at 1 and let iC2 range from 0 to about .128 for 50 frames; then reverse the order for another 50 frames to get a ping-pong effect. I use Pictsaver to save each frame. At several iC2 values (.109, .113, etc.) you'll see that tendrils from various objects touch. This animation is about 1.7MB and takes three disks; it plays with the PD program *ShowAnim* and requires 3MB of memory. I also have a reduced picture size version taking up about 600,000 bytes. If you're interested, send me one or three formatted disks with a stamped return mailer and I'll make you a copy of the animation along with the animation player; no other programs are required to view the animations.

Next Time

This program is on the magazine disk as SINZ.ASM and SINZ along with all the include files. If you make any changes, assemble the source code using A68K SINZ.ASM and link it using BLINK SINZ.O; both A68K and BLINK are also on the disk. In the next article I'll introduce you to a new assembler and we'll use the floating point registers to turn chaos into a thing of beauty.

COMPLEX FUNCTIONS

```
;COMPLEX FUNCTIONS

getele2  macro      ;<a or b>
   movedp   \1,d0
    expdp           ;exp(\1)
   movedp   d0,d6
   movedp   \1,d0
   negdp
    expdp           ;exp(-\1)
   movedp   d0,d4    ;save here
   movedp   d6,d2
    adddp          ;exp(\1)+exp(-\1)
   movedp   twodp,d2
   divdp
  move.l  d0,e1-v(a5) ;(exp(\1)+exp(-\1))/2
   move.l   d1,e1+4-v(a5)

   movedp   d6,d0    ;exp(\1)
   movedp   d4,d2    ;exp(-\1)
    subdp          ;exp(\1)-exp(-\1)
   movedp   twodp,d2
   divdp
  move.l  d0,e2-v(a5) ;(exp(\1)-exp(-\1))/2
   move.l   d1,e2+4-v(a5)
   endm

sinz    macro     ;(c1+ic2)*sin(a+ib))+(f1+if2)
   getele2  b
   movedp   a,d0
    sindp           ;sin(a)
   movedp   e1,d2
   muldp
  move.l  d0,e1-v(a5) ;sin(a)*e1
  move.l   d1,e1+4-v(a5)

   movedp   a,d0
    cosdp           ;cos(a)
   movedp   e2,d2
   muldp
  move.l  d0,e2-v(a5) ;cos(a)*e2
  move.l   d1,e2+4-v(a5)

   movedp   c2,d2
   muldp
   movedp   d0,d6
   movedp   c1,d0
   movedp   e1,d2
   muldp
   movedp   d6,d2
   subdp
   movedp   f1,d2
   adddp
  move.l  d0,a-v(a5)  ;newa=c1*e1-c2*e2+f1
  move.l   d1,a+4-v(a5)

   movedp   c2,d0
   movedp   e1,d2
   muldp
   movedp   d0,d6
   movedp   c1,d0
   movedp   e2,d2
   muldp
   movedp   d6,d2
   adddp
   movedp   f2,d2
   adddp
  move.l  d0,b-v(a5)  ;newb=c2*e1+c1*e2+if2
   move.l   d1,b+4-v(a5)
   movedp   b,d0     ;check abs(b)
   absdp
   endm

cosz    macro     ;(c1+ic2)*cos(a+ib)+(f1+if2)
   getele2  b
   movedp   a,d0
   cosdp
   movedp   e1,d2
   muldp
   move.l   d0,e1-v(a5)
   move.l   d1,e1+4-v(a5)

   movedp   a,d0
   sindp
   movedp   e2,d2
   muldp
   move.l   d0,e2-v(a5)
   move.l   d1,e2+4-v(a5)

   movedp   c1,d0
   move.l   e1,d2
   muldp
   movedp   d0,d6
   movedp   c2,d0
   movedp   e2,d2
   muldp
   movedp   d6,d2
   adddp
   movedp   f1,d2
   adddp
   move.l   d0,a-v(a5)
   move.l   d1,a+4-v(a5)

   movedp   c1,d0
   movedp   e2,d2
   muldp
   movedp   d0,d6
   movedp   c2,d0
   movedp   e1,d2
   muldp
   movedp   d6,d2
   subdp
   movedp   f2,d2
   adddp
   move.l   d0,b-v(a5)
   move.l   d1,b+4-v(a5)
   movedp   b,d0
   absdp
   endm

sinhz   macro     ;(c1+ic2)*sinh(a+ib)+(f1+if2)
   getele2  a
   movedp   b,d0
   cosdp
   movedp   e2,d2
   muldp
   move.l   d0,e2-v(a5)
   move.l   d1,e2+4-v(a5)

   movedp   b,d0
   sindp
   movedp   e1,d2
   muldp
   move.l   d0,e1-v(a5)
   move.l   d1,e1+4-v(a5)

   movedp   c2,d0
   movedp   e1,d2
```

```
   muldp
   movedp   d0,d6
   movedp   c1,d0
   movedp   e2,d2
   muldp
   movedp   d6,d2
   subdp
   movedp   f1,d2
   adddp
   move.l   d0,a-v(a5)
   move.l   d1,a+4-v(a5)

   movedp   c2,d0
   movedp   e2,d2
   muldp
   movedp   d0,d6
   movedp   c1,d0
   movedp   e1,d2
   muldp
   movedp   d6,d2
   adddp
   movedp   f2,d2
   adddp
   move.l   d0,b-v(a5)
   move.l   d1,b+4-v(a5)
   movedp   a,d0
   absdp
   endm

coshz   macro     ;(c1+ic2)*cosh(a+ib)+(f1+if2)
   getelе2  a
   movedp   b,d0
   cosdp             ;cos(b)
   movedp   e1,d2
   muldp             ;cos(b)*e1
   move.l   d0,e1-v(a5)
   move.l   d1,e1+4-v(a5)

   movedp   b,d0
   sindp
   movedp   e2,d2
   muldp
   move.l   d0,e2-v(a5) ;sin(b)*e2
   move.l   d1,e2+4-v(a5)

   movedp   c2,d2
   muldp
   movedp   d0,d6
   movedp   c1,d0
   movedp   e1,d2
   muldp
   movedp   d6,d2
   subdp
   movedp   f1,d2
   adddp
   move.l   d0,a-v(a5) ;newa=c1*e1-c2*e2+f1
   move.l   d1,a+4-v(a5)
   movedp   c2,d0
   movedp   e1,d2
   muldp
   movedp   d0,d6
   movedp   c1,d0
   movedp   e2,d2
   muldp
   movedp   d6,d2
   adddp
   movedp   f2,d2
   adddp
   move.l   d0,b-v(a5) ;newb=c2*e1+c1*e2+if1
   move.l   d1,b+4-v(a5)
   movedp   a,d0       ;check abs(a)
   absdp
   endm

expz    macro     ;(c1+ic2)*exp(a+ib)+(f1+if2)
   movedp   a,d0
   expdp
   movedp   d0,d6    ;exp(a)
   movedp   b,d0
   cosdp
   movedp   d6,d2
   muldp
   movedp   e1       ;exp(a)*cos(b)

   movedp   b,d0
   sindp
   movedp   d6,d2
   muldp
   movedp   e2       ;exp(a)*sin(b)

   movedp   c2,d2
   muldp             ;c2*e2
   movedp   d0,d6
   movedp   c1,d0
   movedp   e1,d2
   muldp             ;c1*e1
   movedp   d6,d2
   subdp
   movedp   f1,d2
   adddp
   move.l   d0,a-v(a5) ;c1*e1-c2*e2+f1
   move.l   d1,a+4-v(a5)

   movedp   c2,d0
   movedp   e1,d2
   muldp             ;c2*e1
   movedp   d0,d6
   movedp   c1,d0
   movedp   e2,d2
   muldp             ;c1*e2
   movedp   d6,d2
   adddp
   movedp   f2,d2
   adddp
   move.l   d0,b-v(a5) ;c2*e1+c1*e2+f2
   move.l   d1,b+4-v(a5)
   movedp   a,d0     ;check a
   endm
```

Listing 2

```
;LISTING 2
;  by William P. Nee
;     Rt 2, Box 216C
;     Mason WI  54856-9302
;SIN(Z) or COS(Z) in double precision
   jmp   start
   include execmacros.i
   include intmacros.i
   include dosmacros.i
   include gfxmacros.i
   include dpmathmacros.i
   include complex.i
   include menu.i

count equr a2
across  equr a3
```

```
down    equr  a4
depth equ 5

@pset  macro        ;<across,down>
  movea.l  rp(pc),a1
   move.w   \1,d0
   add.l    xoffset(pc),d0          ;adjust across
   move.l   yoffset(pc),d1          ;adjust down
   sub.w    \2,d1
   ext.l    d0
   ext.l    d1
  move.l   gfxbase(pc),a6
   jsr      -324(a6)
   endm
@cfm1   macro   ;(branch to if no msg)
  movea.l  window(pc),a0
  move.l   #menunull,d0
   intlib   onmenu
  movea.l  userport(pc),a0
   syslib   getmsg
   tst.l    d0
   beq      \1
  movea.l  d0,a1
  move.l   im.class(a1),d2   ;IDCMP
  move.w   im.code(a1),d3    ;menu#
   syslib   replymsg
   endm
@cfm2    macro   ;(branch to if no msg)
  movea.l  userport(pc),a0
   syslib   getmsg
   tst.l    d0
   beq      \1
  movea.l  d0,a1
  move.l   im.class(a1),d2   ;IDCMP
  move.w   im.code(a1),d3    ;ASCII
  move.w  im.mousex(a1),d5  ;Xcoordinate
  move.w  im.mousey(a1),d6  ;Ycoordinate
   syslib   replymsg
   endm

start
   move.l sp,stack             ;save stack pointer

open_libs                      ;open all the libraries
we need
   openlib    int,done
   openlib    dos,close_int
   openlib    gfx,close_dos
   openlib    dpmath,close_gfx
   openlib    dptrans,close_gfx
   doslib     output
   move.l     d0,conhandler
;  jmp setup
options
   cls
   linefeed
   right    31
   boldface
   print    titlemsg
   normal
   linefeed
   print    xmsg
   linefeed
   print    palmsg
   linefeed
   print    cmsg
   linefeed
   print    zoommsg
   linefeed
   linefeed
   right    24
   style    italics,orange,black
   print    lmbmsg
   normal
   linefeed
loop
   lmb      loop
setup:          ;open a screen of 320 x 200
make_screen:
  openscreen myscreen,close_libs
make_window
  openwindow mywindow,close_screen
   mode        jam1
   palette    colormap1(pc),32
   openmenu    menu0
msg_check
   @cfm1    msg_check
   cmpi.l   #menupick,d2
   bne.s    msg_check
  eval_menunumber
   tst.w    d0
   bne.s    msg_check
   move.w   d1,type
   closemenu
dp_newton_demo
get_values
  removegadgets
   lea     gadget1buffer(pc),a0
   bsr      convertdp
   movedp   c1

   lea     gadget2buffer(pc),a0
   bsr      convertdp
   movedp   c2

   lea     gadget3buffer(pc),a0
   bsr      convertdp
   movedp   f1

   lea     gadget4buffer(pc),a0
   bsr      convertdp
   movedp   f2

   lea     gadget5buffer(pc),a0
   bsr      convertdp
   movedp   xleft

   lea     gadget6buffer(pc),a0
   bsr      convertdp
   movedp   ytop

   lea     gadget7buffer(pc),a0
   bsr      convertdp
   movedp   xright

   lea     gadget8buffer(pc),a0
   bsr      convertdp
   movedp   ybottom
   movedp   saveybottom
   bra      requester2  ;draw SMALL or LARGE
scale
   move.l   xlength,d0
   fltdp
   movedp   d0,d6
   movedp   xright,d0
   movedp   xleft,d2
   subdp
   movedp   d6,d2
   divdp
   movedp   xinc              ;x scale

  movedp  saveybottom,ybottom
   move.l   ylength,d0
```

```
   fltdp
   movedp   d0,d6
   movedp   ytop,d0
   movedp   ybottom,d2
   subdp
   movedp   d6,d2
   divdp
   movedp   yinc              ;y scale
   bra      showit

convertdp
   moveq.l  #0,d0
   moveq.l  #0,d1
   moveq.l  #0,d4             ;sign register
   moveq.l  #0,d5
   moveq.l  #0,d7             ;# characters right of
decimal
   suba.l   a2,a2             ;clear a2; decimal flag
register
   cmpi.b   #'-',(a0)         ;a minus sign?
   bne.s    positive
   bset     #31,d4            ;if so set last bit in d4
   addq.l   #1,a0             ;move over one space
positive
getdigit
   move.b   (a0)+,d5          ;next value to d5
   cmpi.b   #'.',d5           ;a decimal?
   bne.s    testdigit
   move.w   #1,a2             ;if so, flag a2
   clr.l    d7                ; and clear d7
   bra.s    getdigit
testdigit
   cmpi.b   #'9',d5           ;above '9' ?
   bhi.s    zero_check        ;branch if so
   cmpi.b   #'0',d5           ;below '0' ?
   blt.s    zero_check        ;branch if so
   andi.l   #$0f,d5           ;convert ASCII to decimal

   movedp   d0,d2             ;must multiply d0 * 10
   asl.l    #1,d1             ;d0 * 2
   roxl.l   #1,d0
   asl.l    #1,d3             ;d2 * 8
   roxl.l   #1,d2
   asl.l    #1,d3
   roxl.l   #1,d2
   asl.l    #1,d3
   roxl.l   #1,d2
   moveq.l  #0,d6
   add.l    d3,d1             ;d0 = d0 + d2
   addx.l   d2,d0
   add.l    d5,d1             ; + this digit
   addx.l   d6,d0

   addq.w   #1,d7             ;number of characters done
   cmpi.w   #16,d7            ;up to 15 ?
   bne      getdigit
dperror
   ;moveq    #1,d1            ;optional error flag
   ;rts
   bra      close_window      ;error - close everything!
zero_check
   movea.l  a0,a5             ;return string location
   tst.l    d1                ;don't try to convert 0
   bne.s    get_exponent      ;branch if not 0
   tst.l    d0                ;check second half
   beq.s    dp_done           ;branch if value is 0
get_exponent
   move.l   #$43f,d6          ;maximum exponent
1$
   subq.l   #1,d6             ;decrease exponent
   asl.l    #1,d1             ;d0 * 2
   roxl.l   #1,d0             ;rotate with carry from d1
   bcc.s    1$                ;branch if no carry
   moveq.l  #11,d5            ;move right 12 bits

shift_right
   lsr.l    #1,d0
   roxr.l   #1,d1
   dbra.s   d5,shift_right
adjust_exponent
   swap     d6                ;move to left word
   asl.l    #4,d6             ;move to left end
   or.l     d6,d0             ;put in d0
fraction_check
   cmpa.l   #0,a2             ;any decimal ?
   beq.s    do_sign           ;no
   subq.l   #1,d7             ;decrease number of digits
   bmi.s    do_sign           ;no digits after the
decimal

decimal_adjust
   move.l   #$40240000,d2     ;dp 10
   moveq.l  #0,d3             ; part 2
   divdp
   dbra.s   d7,decimal_adjust ;do for all digits

do_sign
   or.l     d4,d0             ;add sign to d0
dp_done
   moveq.w  #0,d6             ;all is 'ok'
   rts

showit
   movea.l  rp(pc),a1
   pcls
   move.l   #1,alldone        ;completed? flag
   move.w   #0,down
   lea      v,a5
l1
;  movedp   xleft,xx
   move.l   xleft-v(a5),xx-v(a5)
   move.l   xleft+4-v(a5),xx+4-v(a5)
   move.w   #0,across
l2
;  movedp   xx,a
   move.l   xx-v(a5),a-v(a5)
   move.l   xx+4-v(a5),a+4-v(a5)
;  movedp   ybottom,b
   move.l   ybottom-v(a5),b-v(a5)
   move.l   ybottom+4-v(a5),b+4-v(a5)
   move.w   #0,count          ;clear the count
l3
   move.w   type(pc),d0
type0
   cmpi.w   #0,d0
   bne      type1
   sinz
   bra      compare
type1
   cmpi.w   #1,d0
   bne      type2
   cosz
   bra      compare
type2
   cmpi.w   #2,d0
   bne      type3
   sinhz
   bra      compare
type3
   cmpi.w   #3,d0
   bne      type4
   coshz
   bra      compare
type4
```

```
   expz

compare
   fixdp
   cmpi.l   #50,d0
   blo.s    continue
   move.w   count,d0    ;count -> color
   andi.w   #31,d0
   foreground           ;set apen
   @pset    across,down ;pset the point
   bra.s    14
continue
   addq     #1,count    ;increase count
   cmpa.w   #48,count   ;up to maximum yet ?
   bne      13          ;branch if not
14
   adddp    xx,xinc     ;increase xx by xscale
   move.l   d0,xx-v(a5) ;movedp xx,xinc
   move.l   d1,xx+4-v(a5)
check_for_message
   @cfm2    no_message
   cmpi.l   #mousebuttons,d2
   beq      zoom
   tst.l    d3
   beq      no_message
   andi.l   #$0000ffff,d3  ;ASCII value
   cmpi.w   #'1',d3     ;#1
   beq      do_palette1
   cmpi.w   #'2',d3     ;#2
   beq      do_palette2
   cmpi.w   #'3',d3     ;#3
   beq      do_palette3
   cmpi.w   #'4',d3     ;#4
   beq      do_palette4
   cmpi.w   #'5',d3     ;#5
   beq      do_palette5
   cmpi.w   #'6',d3     ;#6
   beq      do_palette6
   cmpi.w   #'7',d3     ;#7
   beq      do_palette7
   cmpi.w   #'8',d3     ;#8
   beq      do_palette8
   cmpi.w   #'9',d3     ;#9
   beq      do_palette9
   cmpi.w   #'0',d3     ;#0
   beq      do_palette0
   cmpi.w   #'c',d3
   beq.s    new_values
   cmpi.w   #'q',d3
   beq      close_window
   cmpi.w   #'x',d3
   beq.s    redraw_it
   bra      no_message
redraw_it
   eori.l   #1,redraw
   beq      draw_small
   bra      draw_large
new_values
   pcls
   closewindow
   bra      make_window
do_palette1
   palette colormap1(pc),32
   bra      no_message
do_palette2
   palette colormap2(pc),32
   bra      no_message
do_palette3
   palette colormap3(pc),32
   bra      no_message
do_palette4
   palette colormap4(pc),32
   bra      no_message
do_palette5
   palette colormap5(pc),32
   bra      no_message
do_palette6
   palette colormap6(pc),32
   bra      no_message
do_palette7
   palette colormap7(pc),32
   bra      no_message
do_palette8
   palette colormap8(pc),32
   bra      no_message
do_palette9
   palette colormap9(pc),32
   bra      no_message
do_palette0
   palette colormap0(pc),32
   bra      no_message
zoom
   andi.l   #$0000ffff,d5  ;mouse x
   andi.l   #$0000ffff,d6  ;mouse y
   move.l   d5,startx
   move.l   d6,starty
   mode     complement
lmb_down
   @cfm2    lmb1
   cmpi.w   #selectup,d3
   beq      lmb_up
lmb1
   andi.l   #$0000ffff,d5  ;mouse x
   andi.l   #$0000ffff,d6  ;mouse y
   move.l   d5,endx
   move.l   d6,endy
   box      startx,starty,endx,endy,21
   delay    1
   box      startx,starty,endx,endy,21
   bra      lmb_down
lmb_up
   box      startx,starty,endx,endy,21
   mode     jam1
requester1
   movea.l  window(pc),a0
   lea      body1text,a1
   lea      positivetext,a2
   lea      negativetext,a3
   moveq    #0,d0
   moveq    #0,d1
   move.l   #130,d2
   move.l   #50,d3
   intlib   autorequest
   tst.l    d0
   bne      new_coordinates
   mode     complement
   box      startx,starty,endx,endy,21
   mode     jam1
   tst.l    alldone
   beq      now_what ;finished drawing
   bra      check_for_message
new_coordinates
check_them
   move.l   startx,d0
   move.l   endx,d1
   cmp.l    d1,d0
   blo.s    nc1   ;startx < endx
   exg      d1,d0
   move.l   d0,startx
   move.l   d1,endx
nc1
   move.l   starty,d0
   move.l   endy,d1
   cmp.l    d1,d0
```

```
   blo.s    nc2   ;starty < endy
   exg      d0,d1
   move.l   d0,starty
   move.l   d1,endy

nc2
   move.l   startx,d0
   sub.l    normalizex,d0
   fltdp
   movedp   xinc,d2
   muldp
   adddp    xleft
   movedp   newxleft
   move.l   endx,d0
   sub.l    normalizex,d0
   fltdp
   movedp   xinc,d2
   muldp
   adddp    xleft
   movedp   xright
   movedp   newxleft,xleft

   move.l   starty,d0
   sub.l    normalizey,d0
   fltdp
   movedp   yinc,d2
   muldp
   movedp   d0,d2
   movedp   ytop,d0
   subdp
   movedp   newytop
   move.l   endy,d0
   sub.l    normalizey,d0
   fltdp
   movedp   yinc,d2
   muldp
   movedp   d0,d2
   movedp   ytop,d0
   subdp
   movedp   ybottom
   movedp   saveybottom
   movedp   newytop,ytop

requester2
   movea.l  window(pc),a0
   lea      body2text,a1
   lea      smalltext,a2
   lea      largetext,a3
   moveq    #0,d0
   moveq    #0,d1
   move.l   #130,d2
   move.l   #50,d3
   intlib   autorequest
   tst.l    d0
   beq      draw_large
draw_small
   move.l   #128,xlength
   move.l   #129,xlength1
   move.l   #128,ylength
   move.l   #129,ylength1
   move.l   #96,xoffset
   move.l   #164,yoffset
   move.l   #96,normalizex
   move.l   #36,normalizey
   move.l   #0,redraw
   bra      scale
draw_large
   move.l   #320,xlength
   move.l   #320,xlength1
   move.l   #200,ylength
   move.l   #200,ylength1
   move.l   #0,xoffset
   move.l   #199,yoffset
   move.l   #0,normalizex
   move.l   #0,normalizey
   move.l   #1,redraw
   bra      scale
no_message
   add.w    #1,across          ;across one space
   cmpa.l   xlength1,across    ;all way across yet ?
   bne      l2                 ;branch if not

   adddp    ybottom,yinc       ;increase yy by yscale
   movedp   ybottom
   add.w    #1,down            ;down one space
   cmpa.l   ylength1,down      ;all way down yet ?
   bne      l1                 ;branch if not
   move.l   #0,alldone         ;finished drawing

now_what
   @cfm2    now_what_done
   cmpi.l   #mousebuttons,d2
   beq      zoom
   andi.l   #$0000ffff,d3
   cmpi.w   #'1',d3     :#1
   beq.s    do_palette11
   cmpi.w   #'2',d3     ;#2
   beq      do_palette21
   cmpi.w   #'3',d3     ;#3
   beq      do_palette31
   cmpi.w   #'4',d3     ;#4
   beq      do_palette41
   cmpi.w   #'5',d3     ;#5
   beq      do_palette51
   cmpi.w   #'6',d3     ;#6
   beq      do_palette61
   cmpi.w   #'7',d3     ;#7
   beq      do_palette71
   cmpi.w   #'8',d3     ;#8
   beq      do_palette81
   cmpi.w   #'9',d3     ;#9
   beq      do_palette91
   cmpi.w   #'0',d3     ;#0
   beq      do_palette01
   cmpi.w   #'c',d3
   beq      new_values
   cmpi.w   #'x',d3
   beq      redraw_it
   cmpi.w   #'q',d3
   beq      close_window
now_what_done
   bra      now_what

do_palette11
   palette  colormap1(pc),32
   bra      now_what
do_palette21
   palette  colormap2(pc),32
   bra      now_what
do_palette31
   palette  colormap3(pc),32
   bra      now_what
do_palette41
   palette  colormap4(pc),32
   bra      now_what
do_palette51
   palette  colormap5(pc),32
   bra      now_what
do_palette61
   palette  colormap6(pc),32
   bra      now_what
do_palette71
   palette  colormap7(pc),32
   bra      now_what
```

```
do_palette81
  palette colormap8(pc),32
   bra      now_what
do_palette91
  palette colormap9(pc),32
   bra      now_what
do_palette01
  palette colormap0(pc),32
   bra      now_what
close_window
  closemenu
  closewindow
close_screen
  closescreen
close_libs
  closelib dpmath
  closelib dptrans
close_gfx
  closelib gfx
close_dos
  closelib dos
close_int
  closelib int
done
  move.l   stack,sp
   rts

   evenpc

stack      dc.l  0          ;reserve storage locations
gfxbase    dc.l  0
intbase    dc.l  0
dosbase    dc.l  0
dpmathbase dc.l  0
dptransbase dc.l 0
                            ;library names
gfx        dc.b  'graphics.library',0
   evenpc
int        dc.b  'intuition.library',0
   evenpc
dos        dc.b  'dos.library',0
   evenpc
dpmath     dc.b 'mathieeedoubbas.library',0
   evenpc
dptrans    dc.b 'mathieeedoubtrans.library',0
   evenpc
myscreen:
   dc.w 0,0,320,200,depth     ;depth is 5
   dc.b 0,1
   dc.w 0
  dc.w customscreen
   dc.l 0,0,0,0
   evenpc
mywindow
   dc.w 0,0,320,200
   dc.b 0,1
  dc.l mousebuttons!mousemove!vanillakey!menupick
  dc.l borderless!activate!mousereport!smartrefresh
   dc.l gadget1,0
    dc.l 0
    dc.l 0,0
   dc.w 0,0,0,0
   dc.w customscreen

   evenpc
v:
xleft    dc.l  0,0          ;reserve space for both
parts
xright   dc.l  0,0
xx       dc.l  0,0
xinc     dc.l  0,0
ybottom  dc.l  0,0
ytop     dc.l  0,0
yinc     dc.l  0,0
startx   dc.l  0
starty   dc.l  0
endx     dc.l  0
endy     dc.l  0
xlength  dc.l  0
xlength1 dc.l  0
xoffset  dc.l  0
normalizex dc.l  0
ylength  dc.l  0
ylength1 dc.l  0
yoffset  dc.l  0
normalizey dc.l  0
newxleft dc.l  0,0
newytop  dc.l  0,0
saveybottom dc.l 0,0
alldone  dc.l  0
redraw   dc.l  0
e1       dc.l  0,0
e2       dc.l  0,0
c1       dc.l  0,0
c2       dc.l  0,0
f1       dc.l  0,0
f2       dc.l  0,0
a        dc.l  0,0
b        dc.l  0,0
twodp    dc.l  $40000000,0
type     dc.w  0
   evenpc
colormap1                    ;my new palette
  dc.w $000,$f0f,$d0f,$b0f,$90f,$70f,$50f,$30f
  dc.w $10f,$00f,$03d,$05b,$079,$097,$0b5,$0d3
  dc.w $0f0,$3f0,$6f0,$9f0,$bf0,$df0,$ff0,$fe0
  dc.w $fd0,$fc0,$fa0,$f80,$f60,$f40,$f20,$f00
   evenpc
colormap2
  dc.w $000,$f00,$f20,$f40,$f60,$f80,$fa0,$fb0
  dc.w $fc0,$fd0,$fe0,$ff0,$df0,$af0,$6f0,$3f0
  dc.w $0f0,$0d3,$0a6,$06a,$03d,$00f,$30d,$60a
  dc.w $a06,$d03,$f0f,$d0d,$a0a,$606,$303,$202
   evenpc
colormap3
  dc.w $000,$f00,$d00,$b00,$a00,$800,$600,$500
  dc.w $300,$200,$100,$0f0,$0d0,$0b0,$0a0,$080
  dc.w $060,$050,$030,$020,$010,$00f,$00d,$00b
  dc.w $00a,$008,$006,$005,$004,$003,$002,$001
   evenpc
colormap4
  dc.w $000,$f00,$e00,$d00,$c00,$b00,$a00,$900
  dc.w $800,$700,$600,$0f0,$0e0,$0d0,$0c0,$0b0
  dc.w $0a0,$090,$080,$070,$060,$00f,$00e,$00d
  dc.w $00c,$00b,$00a,$009,$008,$007,$006,$005
   evenpc
colormap5
  dc.w $000,$f00,$e20,$d40,$c60,$b80,$aa0,$8b0
  dc.w $6c0,$4d0,$2e0,$0f0,$0e2,$0d4,$0c6,$0b8
  dc.w $0aa,$08b,$06c,$04d,$02e,$00f,$00e,$00d
  dc.w $00c,$00b,$00a,$009,$007,$005,$003,$001
   evenpc
colormap6
  dc.w $000,$400,$501,$720,$710,$821,$831,$841
  dc.w $951,$a61,$a71,$a81,$b91,$ba1,$bb2,$cc2
  dc.w $ac2,$9c2,$7c2,$6c2,$4d2,$2d1,$1d2,$1d4
  dc.w $1e5,$1e7,$1ea,$1ec,$1ee,$0df,$0bf,$09f
   evenpc
colormap7
  dc.w $000,$620,$730,$951,$a72,$c93,$db4,$fd5
  dc.w $ee4,$cd3,$9c2,$7a2,$591,$481,$270,$150
  dc.w $062,$074,$067,$047,$008,$308,$709,$907
  dc.w $a05,$a13,$b11,$b43,$c72,$da3,$dd3,$be4
   evenpc
```

```
colormap8
  dc.w $000,$fff,$ecd,$c9c,$b7a,$a59,$837,$726
  dc.w $559,$66a,$77b,$99b,$aac,$ccd,$dde,$fff
  dc.w $ded,$cdb,$bc9,$9b7,$8a6,$794,$685,$572
  dc.w $58a,$69b,$7ab,$9bc,$acd,$bcc,$cee,$fff
   evenpc
colormap9
  dc.w $000,$fe0,$db0,$c90,$a60,$950,$730,$620
  dc.w $fe0,$af0,$3f0,$0f3,$0fa,$0df,$07f,$00f
  dc.w $fe0,$eb0,$c90,$b90,$a60,$940,$730,$620
  dc.w $fe0,$de0,$ad0,$7c0,$5b0,$3a0,$270,$080
   evenpc
colormap0
  dc.w $000,$004,$008,$00b,$00f,$00b,$008,$004
  dc.w $000,$430,$770,$ba0,$fe0,$ba0,$860,$430
  dc.w $000,$300,$700,$a00,$e00,$a00,$700,$300
  dc.w $000,$020,$040,$060,$080,$060,$040,$020
   evenpc
menus
  makemenu menu0,'Project',,0,1
   makeitem menu0item0,'Sin(Z)',menu0item1,0,$153,$1e
   makeitem menu0item1,'Cos(Z)',menu0item2,10,$53,$1d
   makeitem menu0item2,'Sinh(Z)',menu0item3,20,$53,$1b
   makeitem menu0item3,'Cosh(Z)',menu0item4,30,$53,$17
   makeitem menu0item4,'Exp(Z)',,40,$53,$f
   evenpc
messages
titlemsg dc.b 'COMPLEX ITERATIONS',0
   evenpc
cmsg    dc.b  ' press c for main menu',0
   evenpc
xmsg    dc.b  ' press x to switch picture size',0
   evenpc
palmsg  dc.b  ' press 0 - 9 to change palettes',0
   evenpc
zoommsg dc.b  ' use the LMB to draw a ZOOM box',0
   evenpc
lmbmsg  dc.b 'NOW PRESS THE LEFT MOUSE BUTTON',0
   evenpc
gadgets
  makestrgadget
gadget1,'C1',gadget2,40,75,100,9,$0,$1,$0,1,15
  makestrgadget
gadget2,'iC2',gadget3,180,75,100,9,$0,$1,$0,2,15
  makestrgadget
gadget3,'F1',gadget4,40,100,100,9,$0,$1,$0,3,15
  makestrgadget
gadget4,'iF2',gadget5,180,100,100,9,$0,$1,$0,4,15
  makestrgadget
gadget5,'Xleft',gadget6,40,150,100,9,$0,$1,$0,5,15
  makestrgadget
gadget6,'Ytop',gadget7,110,125,100,9,$0,$1,$0,6,15
  makestrgadget
gadget7,'Xright',gadget8,180,150,100,9,$0,$1,$0,6,15
  makestrgadget
gadget8,'Ybot',0,110,175,100,9,$0,$1,$0,8,15
   evenpc
gadget1buffer dc.b '1.00000000000',0 ;C1
   evenpc
gadget2buffer dc.b '0.00000000000',0 ;iC2
   evenpc
gadget3buffer dc.b '0.00000000000',0 ;F1
   evenpc
gadget4buffer dc.b '0.00000000000',0 ;iF2
   evenpc
gadget5buffer dc.b '-4.0000000000',0 ;Xleft
   evenpc
gadget6buffer dc.b '4.50000000000',0 ;Ytop
   evenpc
gadget7buffer dc.b '4.00000000000',0 ;Xright
   evenpc
gadget8buffer dc.b '-4.5000000000',0 ;Ybot
   evenpc

simple_requesters
body1text   ;intui-text structure for body text
   dc.b    0,1,1,0
   dc.w    0,0
   dc.l    0,body1string,0
   evenpc
body1string  dc.b 'OK to ZOOM ?',0
   evenpc
positivetext
   dc.b    0,1,1,0
   dc.w    0,0
   dc.l    0,positivestring,0
   evenpc
positivestring dc.b 'YES',0
   evenpc
negativetext
   dc.b    0,1,1,0
   dc.w    0,0
   dc.l    0,negativestring,0
   evenpc
negativestring dc.b 'NO!',0
   evenpc
body2text
   dc.b    0,1,1,0
   dc.w    0,0
   dc.l    0,body2string,0
   evenpc
body2string dc.b 'What Size ?',0
   evenpc
smalltext
   dc.b    0,1,1,0
   dc.w    0,0
   dc.l    0,smallstring,0
   evenpc
smallstring dc.b 'SMALL',0
   evenpc
largetext
   dc.b    0,1,1,0
   dc.w    0,0
   dc.l    0,largestring,0
   evenpc
largestring dc.b 'LARGE',0
   evenpc
   end
```

✔

The complete set of listings can be found on the AC's TECH disk.

Please write to:
Bill Nee
c/o AC's TECH
P.O. Box 2140
Fall River, MA 02722

Pseudo-Random Number Generation

BY CHRISTOPHER JENNINGS

Since most popular languages include functions to generate sequences of random numbers, such as BASIC's own Rnd() command, an article on methods of random number generation may initially seem pointless. Once the implications of this knowledge are considered, however, it will become obvious that this is not the case. One of the biggest problems with having ready-made functions available is the inevitable loss of power that comes with their implementation. Some of the most useful applications of pseudo-random sequence theory are inaccessible by the programmer because control is hidden behind an intermediate function that conceals all the work and is only designed to use one static set of parameters. However, hand coding custom functions involves slightly more work, but gives precise control over how the routine will behave. Assembly language programmers have no choice; lacking access to any built-in commands, they have to code their own routines.

As you are no doubt aware, it is impossible for a computer to generate a true sequence of random numbers; each new number is calculated from the previous value using a reiterative formula. By far the most widespread algorithm is the Linear Congruential Method, which can be expressed as:

$$R[n] = (A*R[n-1] + C) \bmod M, \quad n = 1, 2, ..., M - 1$$

We begin with a seed value (R[n-1]), which must be set by the program for the first call (when n=1). This seed is then multiplied by a constant integer multiplier (A), and summed with a constant integer increment (C). This total is then divided by a constant integer modulus (M), and the remainder becomes the new random number. When the next number is generated, this value will become the next seed value. If certain restrictions on the constants are followed, the above formula will produce a sequence of numbers of period M such that every integer from 0 to M-1 appears exactly once. The last random number will then regenerate the original seed, and the sequence will repeat (there can only be M remainders when one divides by M). The constant restrictions are as follows:

1) The values of C and M must be relatively prime, or have no common factors.

2) (A - 1) must be a multiple of every prime factor of M.

3) If M is a multiple of 4, (A - 1) must also be a multiple of 4.

Obviously, one wants to choose a very large value for M so that the sequence will continue for as long as possible before it repeats. Be cautious, however, that the value is not so large that it will introduce rounding or overflow errors; this could easily decrease the expected period. This will generally be even more of a consideration if you must simulate the modulo function with code such as:

R = A * R + C - M * Int((A * R + C) / M)

Depending on how the language you use calculates the value of an expression containing a modulo operator, you may also need to be aware of this situation. However, in most cases where such an operator exists, you may feel free to choose as large an M value as your data type allows that fits the requirements above, as long as it won't overflow the datatype you are using (later on in the article, this idea is discussed further).

To change the range of the values given from the default (0 through M-1) to some arbitrary range (0 through X), the following formula is all that is required:

R' = R / (M-1) * X

To generate fractional numbers in the range 0 <= N <= 1 (the type of results returned by most programming languages), simply eliminate the X in the above formula.

Of course, if your program uses the same initial seed value each time it is executed, it will result in the same sequence each time. The usual solution to this problem is to scale the first number from a value based on time, for example, the number of jiffies elapsed since midnight. Since the odds of the user booting a system and starting your application within the exact same number of fiftieths or sixtieths of a second twice in a row are infinitesimal, this works very well.

High Speed Random Numbers

If you are writing an application where speed is more important than mathematical accuracy, you won't want to execute the several instructions involved in the expansion of the above formula. This is of particular importance when you are writing a high speed game or similar software where it is particularly important to squeeze as much speed from the processor as possible. In this case, one very effective method is to call the graphics library function VBeamPos(). The purpose of this function is to return the current vertical position of the beam, as a value from 0 to 511. Since the screen is redrawn 60 times a second (NTSC), and each of these redraws sees the video beam run all 512 positions, this will yield fairly random results, especially if you are multitasking. If you feel you must use the hardware to read these values to avoid the overhead caused by the library call, something which you should consider a last resort reserved for entertainment software, table 1 summarizes the beam position counter registers.

Table 1

Register	Address	Bits	Description
VPOSR	$DFF004	15	Long-frame bit; for interlaced displays
		14-1	Unused
		0	High bit of vertical position (V8). Allows for PAL display counts (313).
VHPOSR	$DFF006	15-8	Low bits of vertical position (V7-V0).
		7-0	Horizontal position (H8-H1).

The addresses given are relative to the MC680x0 processor; within the custom chips themselves, the equivalent addresses are $4 and $6 respectively. These are the read-only registers. The easiest way to generate a number with the hardware is to copy VHPOSR and mask out bits 0-7 (H1-H8) by reading the destination as a byte. This will result in a number in the range 0-255. Execute the VBeamPos program for a graphic example of the type of distribution this method provides.

Finding Values for M, A, C, and R

The only difficult part of implementing a LCM sequence is determining appropriate values for the variables. As an example of one working method, we will show how the values used in the RndCore.asm code were determined.

First one must choose a value for M. The best way to do this is applying some logic to your situation. Once you have what appears to be a workable value, try to come up with some A and C values to fit it. If this doesn't work out, you will probably have to reduce the number. In RndCore, we wanted to use the MC680x0's 16-bit multiplication and division opcodes, and so since we were using the unsigned form of the instructions, the first obvious limit is 2^16, or 65536. However, it is immediately obvious that the number must be smaller than this, or numbers at the high end of the scale will create overflow problems (For example, when R=65500, R' will be greater than 65535. This is true since R' = (A*R+C) mod M, and A, C >1.). So initially we will try 2^15, or 32768. Since we must find the prime factors of M, we begin by constructing a factor tree. As you can see, there is only one unique factor: 2. Although this is obvious, since the value was defined as an exponent of two, this is still a good step to use since other numbers will not be so obvious. Recall that (A-1) must be a multiple of the prime factors of M, and a multiple of four if M is a multiple of four (which it is here). Since four is both a multiple of two and four, we have A-1=4, and A=5. Our selection of C is only limited by the fact that C and M must be relatively prime. Since the only factor of M is two, this creates a very advantageous situation in that C can be any odd number (and therefore our selection of C=12479 was arbitrary). This presents an interesting opportunity for creating new sequences: instead of using a timer to determine the initial seed value, we can change the value of C. For example, suppose we call VBeamPos(). If we then multiply the result by two, we will get an even number regardless of the original parity of the number (since it is now divisible by two). Since every even number

is followed by an odd number, we then simply add one to get an odd value for C, and we have a new order for our random sequence. In general you will find it easiest to choose an M value based on a natural exponent of two where possible. Although other values are certainly possible, and sometimes necessary, exponents of two have several characteristics that make them good candidates: they use the same base as the computer itself, they can always use A=5 (which is a relatively small value and thus allows M to be proportionally larger), and any odd value is valid for C.

As a last note on RndCore, it would seem that with A=5 and M=32768, we would still get a value greater than 2^16 at times, and we do. However, due to the nature of the DIVU instruction, this does not pose the problem one would expect it to. Running the following snippet of code through a monitor will demonstrate this (it can be found on disk as MulDiv.asm):

```
move    #$8000,d0;      Put in 32768
mulu    #$4,d0    ;     x 4 (or 2 x $FFFF: 17 bits)
divu    #$8000,d0;      Divide it out, still okay
move    #$8000,d0;      Replace 32768
mulu    #$8,d0    ;     x 4 (or 4 x $FFFF: 18 bits)
divu    #$8000,d0;      Gives wrong result (overflow)
rts
```

As you will see, the instructions will work correctly with up to 17 bits of data; this means that 32768 is sufficiently low to be safe to use.

Applications

Beyond the obvious, there are special applications of numbers generated using a linear congruential formula that derive from the properties described earlier. For example, suppose you are filling a grid which is 64 x 32 in size with 500 objects of different types, and each object must appear at a unique location. A traditional method would be to compare random selections to the current grid until an empty space is found. In pseudocode, this would appear as follows:

```
Create Array MAP[2048]
Randomize Seed Value Using Timer
Function FILLMAP
    Loop N from 1 to 500
        Repeat
                Let NUM=Rnd(2048)
        Until MAP[ NUM ]  = Empty Location
        Let MAP[ NUM ] = Full Location
    End Loop
End Function
```

If you chose to utilize a linear congruential system with M=2048, you would already know that each number would be unique, and your code could be reduced to the following:

```
Create Array MAP[2048]
Randomize Seed Value Using Timer
Function FILLMAP
    Loop N from 1 to 500
        MAP[ L.C. Rnd Number] = Full Location
    End Loop
End Function
```

Although having to write your own random number routine might seem to make the size of the actual sourcecode larger, remember that the built-in random number function is doing the same thing; the compiled code will be essentially the same size (notice the small size of the Assembly Language routine given in Listing 3). Compare this to the size of the code you would require if the objects consisted of 10 or 20 different types, and a substantial savings appears.

If you have ever played the commercial game *Civilization*, you have seen one of many versions of the "fizzle effect." This effect is created by randomly pasting chunks of a picture from a buffer onto the display, causing the picture to appear piece by piece (execute the included program "fizzle" for a visual demonstration). This effect is easy to recreate using a pseudo random sequence where M is equal to the total number of pieces to be moved. Although using pixel-sized chunks is possible, on lower-end machines with a large picture this effect will take a very long amount of time and a larger size is recommended. If you don't require the pieces to appear in a different order with each run, start with a random seed that you select and use the included LCMLab program to create a table of values and use that to create the display; this will execute slightly faster than calculating it with each run.

Program Listings

The following program listings demonstrate examples of using the techniques in this article in Assembly Language, C, and BASIC; the majority are in BASIC since this is the one language that nearly all programmers comprehend. The first two are tools you can use to experiment with different A, R, C and M values, given in C and BASIC versions. The third is the core of an Assembly Language routine you can modify to suit your particular needs. Listing 4 is intended to be used to test the uniformity of distribution of a given random sequence. Listing 5 is a simple demonstration of the VBeamPos() function as a random number generator, and the last represents code for the aforementioned fizzle effect.

Listing 1: LCMLab.c

```
/*       Listing 1: LCMLab.c
         Try M=32768, A=5, C=12131, R=0
   Compiled with SAS/C 6.x */

#include <stdio.h>
main()
{
       long M,A,C,R;
       int  NumValues,Counter;

      printf ("Linear Congruential Pseudo-Random\n");
       printf ("      Sequence Laboratory\n\n");
       printf ("     Copyright (C) 1993 by\n");
       printf ("     Christopher Jennings.\n");
```

```
        printf ("      All Rights Reserved\n\n");
       printf ("Enter modulus M=0 to exit.\n\n");
        for(;;)
        {
              printf ("Enter a modulus    (M): ");
              scanf  ("%ld",&M);
              if (M == 0)
                     return 0L;
             printf ("Enter a multiplier (A): ");
              scanf  ("%ld",&A);
             printf ("Enter an increment (C): ");
              scanf  ("%ld",&C);
             printf ("Enter a seed value (R): ");
              scanf  ("%ld",&R);
             printf ("\nNumber of values to generate?
");
              scanf  ("%d",&NumValues);
              /* Generate the values */
             printf ("%ld ",R);
            for(Counter=1; Counter++ <= NumValues;)
              {
                     R = (A * R + C) % M;
                    printf ("%ld ",R);
              }
              printf ("\n\n");
        }
}
```

Listing 2: LCMLab.bas

```
' Listing 2: LCMLab.bas
' Try M=100, A=21, C=7, R=0
Cls
Print "Linear Congruential Pseudo-Random"
Print "      Sequence Laboratory"
Print
Print "     Copyright (C) 1993 by"
Print "    Christopher Jennings."
Print "      All Rights Reserved"
Print
Print "Enter modulus M=0 to exit."
Do
   Print
   Input "Enter a modulus    (M): ";M
   If M=0 Then Exit
   Input "Enter a multiplier (A): ";A
   Input "Enter an increment (C): ";C
   Input "Enter a seed value (R): ";R
   Print
   Input "Number of values to generate? ";NUM_VALUES
   ' Generate the series of values...
   Print R;
   For COUNTER=1 To NUM_VALUES
      R=(A*R+C) mod M
      Print R;
   Next
   Print
Loop
```

Listing 3: RndCore.asm

```
; Listing 3:
; Assembly Language
; Random Number Core Routine
; ————————
; Copyright © 1993 by
; Christopher Jennings
; All Rights Reserved

; GenRnd:
; Given a 16-bit unsigned seed value < 32768
; in d7, returns the next random number.
; Uses M=32768, A=5, C=12479
; Computes RPN form of R'=(A*R+C)%M

GenRnd:
    mulu    #5,d7
    add     #12479,d7     ; Any odd number works here
    divu    #32768,d7
    clr     d7            ; Same as using ANDI.L $FFFF,D7
after
    swap    d7            ; SWAP D7, but faster & 4 bytes
shorter
    rts
```

Listing 4: DisTest.bas

```
' Listing 5: DisTest.bas
Dim TRACK(9)
Cls
Print "Uniformity of Distribution Test"
Print
Print "    Copyright (C) 1993 by"
Print "   Christopher Jennings."
Print "     All Rights Reserved"
Print
Print "Enter modulus M=0 to exit."
Do
   Print
   Input "Enter a modulus    (M): ";M
   If M=0 Then Exit
   Input "Enter a multiplier (A): ";A
   Input "Enter an increment (C): ";C
   Input "Enter a seed value (R): ";R
   Print
   Input "Number of values to track for test?
";NUM_VALUES
   Print
  Print "Computing distribution..."
  For COUNTER=1 To NUM_VALUES
     R=(A*R+C) mod M
     N=Int(R/M*10)
    TRACK(N)=TRACK(N)+1
   Next
   Print
   Print "Range              % Distribution"
```

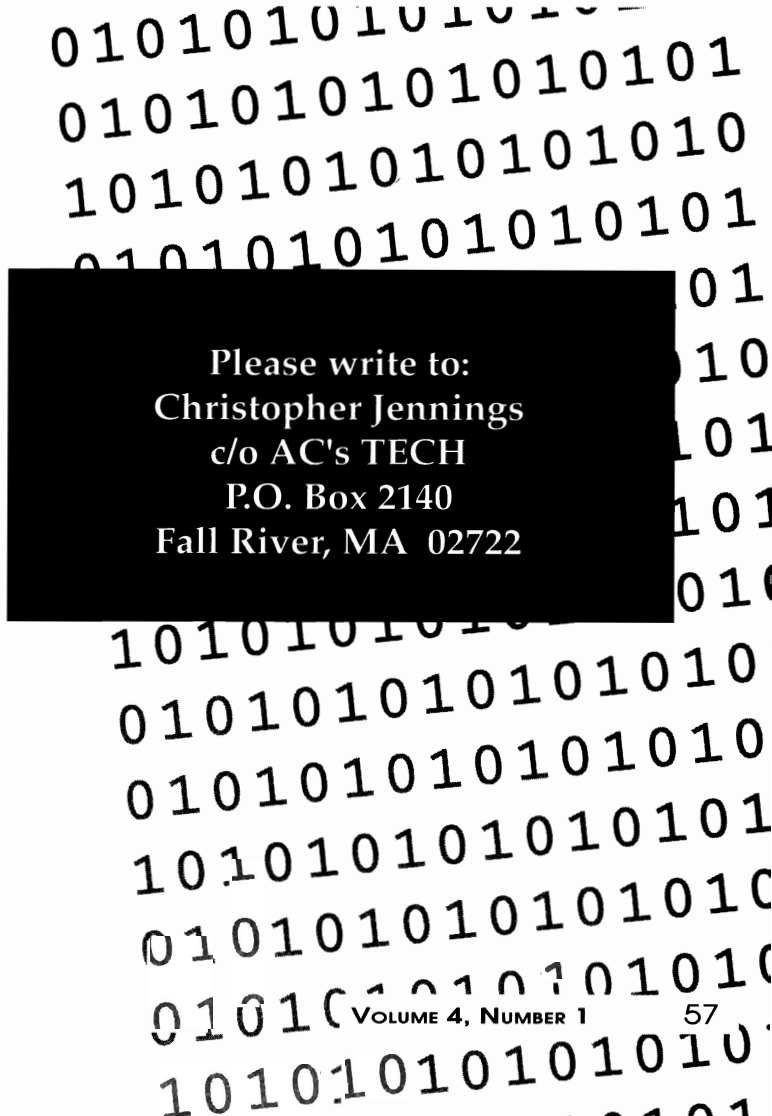
```
  AVG=0
  For COUNTER=0 To 9
    Print COUNTER/10.0;" <= X < ";(COUNTER+1)/10.0;
   TRACK(COUNTER)=TRACK(COUNTER)/NUM_VALUES*100
    AVG=AVG+TRACK(COUNTER)
     Print "   %";TRACK(COUNTER)
  Next
  Sort TRACK()
   Print
  Print "Average: %";AVG/10
  Print "Range  : %";TRACK(9)-TRACK(0)
   Print
Loop
```

Listing 5: VBeamPos.bas

```
' Listing 5: VBeamPos.bas
' Directly reading V7-V0 through hardware
' is the fastest implementation (yielding
' a number from 0 to 255), but this is
' the safest.

'
' Draws a graph on the screen that shows
' distribution of the numbers.

Dim track(511)Screen 0,2,h+1,"VBeamPos() Distribution -
Press mouse button to exit"

Do
 a=VBeamPos()
  Inc track(a)
 If track(a)<389 Then Plot a,400-track(a),1
  If Mouseb<>0 Then End
Loop
```

Listing 6: Fizzle.bas

```
; Listing 6: Fizzle.bas
;
; Example of the fizzle effect
col=0
BitMap 1,320,200,3  ; Define a work region for 320x200x8
Screen 0,3,"Fizzle Effect Demo: Mouse button quits"
For t=1 To 8
 Colour t,t*1.9,t*0.3,t*1.5
Next
Colour 0,0,0,0
Do
  ; Draw the next frame in the buffer
  BitMap 1
  For t=0 To 319
   tmp=col MOD 8+1
   Line 159,99,t,0,tmp
   Line 159,99,319-t,199,tmp
    Inc col
    Gosub CheckButton
  Next
  For t=0 To 199
   tmp=col MOD 8+1
   Line 159,99,319,t,tmp
   Line 159,99,0,199-t,tmp
    Inc col
    Gosub CheckButton
  Next
  col=col+5
  ; Fizzle the frame onto the main screen
  ; Uses 1000 8x8 chunks, M=1000, A=21, C=3, R=0
  r=0
  For t=1 To 1000
    x=r MOD 40*8
    y=r/40*8
    GetShape 0,x,y,8,8  ; Grab an 8x8 block at (x,y)
     BitMap 0
     PutShape 0,x,y      ; Paste it on the visible screen
     BitMap 1
     r=(21*r+3) MOD 1000
     Gosub CheckButton:
  Next
Loop
CheckButton:
 If Mouseb(0)<>0 Then End
 Return
```



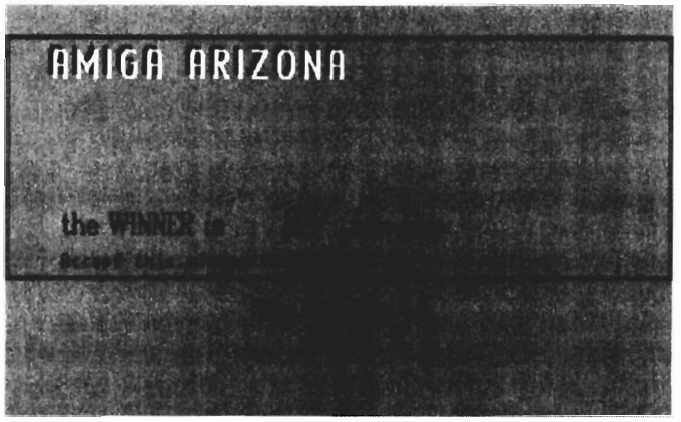
Please write to:
Christopher Jennings
c/o AC's TECH
P.O. Box 2140
Fall River, MA 02722

Draw5.0

by T. Darrel Westbrook

AMOS Professional Door Prize Selection

In this article I will explore AMOS Professional programming and how I used it to solve a practical programming problem. It should help new AMOS Professional users with limited programming experience get a start using this graphics BASIC language. I will describe one way to get the Amiga system date into an AMOS Professional program, a technique that ensures program files are available for program use, a method to read and write text files within AMOS Professional, and an easy way to create a BOB and Icon bank.



In this article, AMOS Professional functions and commands are in bold print, variables capitalized and italicized, AmigaDOS commands italicized, and procedure names capitalized. If you type in this program, do not type in the line numbers. They are for reference only.

It is difficult to learn a new programming language, especially one that has as many commands and functions as AMOS Professional. I believe the best way to learn a new language is by using it. I was looking for some worthwhile project to explore my new AMOS Pro Version 2.0, when my father-in-law presented me with a real-world problem that needed a programming solution.

My father-in-law is the librarian for a large Amiga Users Group in Arizona. Their executive officers, who bring their Amiga computers to club meetings to show software and programming techniques, recently purchased A4000s. They hold a drawing for the members present at the meeting for door prizes supplied by the club or local vendors. In the past, they used an AmigaBASIC program that originally appeared in *Amazing Computing*, volume 3.2, pages 60 and 64. However, that they problems running the program, called *Draw4.0*, on the newer machines. My father-in-law asked if I could help with the problem by converting the AmigaBASIC program to AMOS Pro. The original program was five pages long and I didn't believe it would be much trouble to convert it. But, as I got into the programming, I determined the conversion would take longer than I had originally anticipated. Before the programming got out of hand, I had to step back from the problem and take another look at it.

The code conversion was not going to work, because I was not using the strengths of AMOS Pro. I was strangling AMOS Pro with the limits imposed by AmigaBASIC. Therefore, I redefined the problem to write a new program with the look and feel of the original, but using the strengths of AMOS Professional.

I had completed the appropriate study of the language manual and I was familiar with the commands and what they can do. At this point, it becomes an issue of bringing all the parts together (the problem, the language, problem solving abilities, and imagination) to create a solution. Notice I said a solution not *the* solution. My solution and your solution to a given problem will never be entirely the same, because our imaginations are not the same. This is why there are many solutions to a given problem.

The intent of the AmigaBASIC program was to demonstrate the use of system fonts within an AmigaBASIC program. It used three different fonts and graphics to create an eye-appealing display. When the program started, it scrolled the words Software Giveaway down the top left hand of the screen. Once the program finished scrolling, the club name, in another font, appeared just above "Software Giveaway." The program then instructed the user to press the left mouse button to select a winner. A text file exported from a database contained the club members' names. See Figure 1 for a representation of the AmigaBASIC program display.

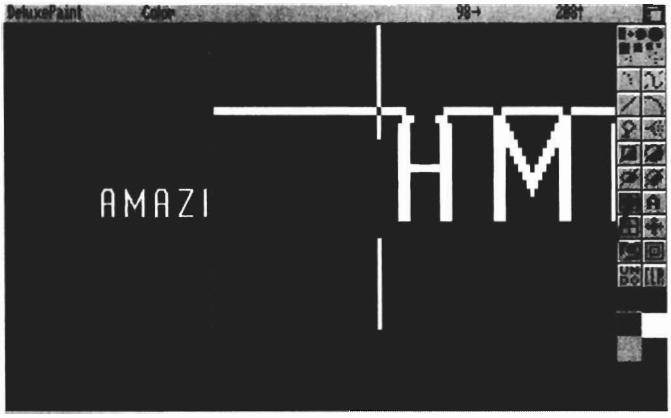
When you pressed the mouse key, the program displayed a winner picked at random and asked the operator to verify if it should accept this as a winner (see Figure 1). If it was not a winner (i.e., the person was not present), the program logged the selected name in a file. Club members used this information to post the results in the club's newsletter. If the selection is a winner, Draw4.0 updates the winner file and then branches back to the selection loop. The program stops when the user either has no more names to select or the user selects 'E' for the name selection.

I initially tried to convert, verbatim, the AmigaBASIC code. I had most of the initialization complete when I hit my first stumbling block. AMOS Pro does not have any date functions or commands to get the computer system date. After searching the user manual in vain for an AMOS date function, I discovered the Exec command.

The Exec command sends a command line to AmigaDOS. It provided an indirect means to obtain the system date.

Study the code for the _GET_SYSTEM_DATE procedure (line 312 to 342 of Listing 1). Through the Exec command, it uses the AmigaDOS date command to redirect the output to a file on the RAM: device named Date_Data.txt. The procedure then opens the file, asks how big it is (the Lof(1), Length of file function), reads it all at once, closes the file, and then deletes it. The remainder of the procedure puts the date in the format the club uses (i.e., MM-DD-YY) and assigns it to the global DATE$ variable. The next problem the program faces is determining if all the required files are available.

The _GET_FILE_INFO procedure assigns Draw: to whatever directory you select if it cannot locate the assigned logical device

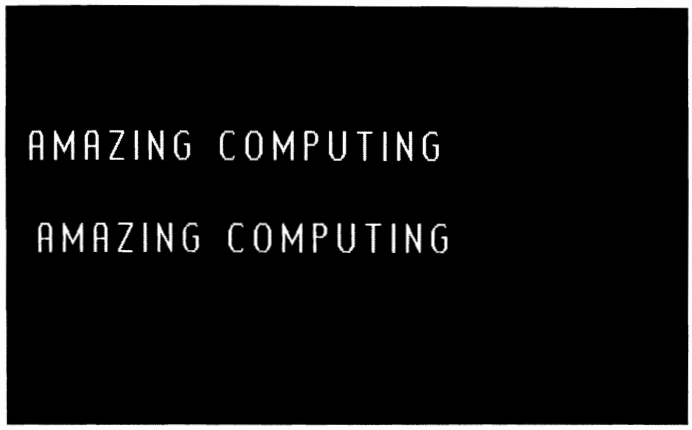


when you start the program. It also finds the location of the members.txt, losers.txt, and winners.txt files. These files are normally in the same directory, but the _GET_FILE_INFO procedure permits each file to be in a separate directory. If it cannot find one of the files in the current directory, it pops up an AMOS requester. You select the location of the file or files. Once Draw5.0 knows the location of the files, it reads the members.txt file into memory.

AMOS Professional has an excellent text display reader (see Read Text command, page 05.07.06 of the AMOS Pro manual), but it is not sizable, and you cannot get the text from the reader that I'm aware of and load it for program manipulation.

The _TEXT_FILE_READ and the _TEXT_FILE_WRITE procedures allow the Draw5.0 program to read in and write to a text file. You can use the _TEXT_FILE_READ procedure to read any text file into an array named LINE$(). The specialized _TEXT_FILE_WRITE procedure writes the information from the program working array called _ARRAY$(,) to a text file. You could easily modify it to handle general text-file writing procedures that would work with any array for loading data into a text file.

The _TEXT_FILE_READ procedure opens the file, FILENAME$, and reads it all at once. The procedure reads the text file into A$, a string variable, which limits the file size Draw5.0 can handle. The buffer size or the string length, whichever is smaller, sets the limits for amount of data the procedure can process. Draw5.0 sets the buffer



size to 64KB (see line 1, Listing 1). The maximum string length is 65000 bytes so the text file limit is roughly 64KB.

After the program reads the text file, it finds and counts the line feeds, which mark the end of each line in the text file (lines 169 to 173). Draw5.0 uses the line feeds as markers to separate the one large string (A$), into the lines of the original file, which the program stores in the LINE$() array. You can now manipulate the text data line by line within the AMOS program.

The _ARRAY$(,) matrix three elements are:

```
_ARRAY$(n,0) - club membership number
_ARRAY$(n,1) - member's name
_ARRAY$(n,2) - flag, set to "-1" when name selected
```

Draw5.0 logs a member's name in the winners.txt or losers.txt text files. The program places a "-1" in element _ARRAY$(n,2) when it selects a name (see _RANDOM procedure, line 129-144). This option prevents the selection of the name as the program continues. The club members use the winner and looser information in the monthly club newsletter. Since the text file is downloaded from a database program using an ASCII export option, it is not necessary to type names into a special file. Draw5.0 has the computer doing most of the work. To exit the program, you select "E."

As you scan the program code, I'm sure you've seen the statement:

```
Load "draw5.0.Abk"
```

(line 7). This file must be in the same directory as Draw5.0. The draw5.0.Abk file contains eight icons, eight blank icons in the background color, and one BOB (Blitter OBject). I used *DeluxePaint III* to make the BOB and the icons, which are all on the same graphics picture. I used a hi-res screen (640 x 200) with eight colors. Figure 2 is a representation of the picture. I achieved the shadow effect by writing each icon twice. I constructed one icon in the color I wanted for the lettering and the second one in black. I cut the colored icon out and laid it over the black icon. I shifted the colored icon right and up to get the shadow effect. Figure 3 is an example of how to achieve the 3-D shadow effect using DeluxePaint.

When I had the icons complete, the most tedious process started. To cut the icons out of the graphics picture, I needed to know

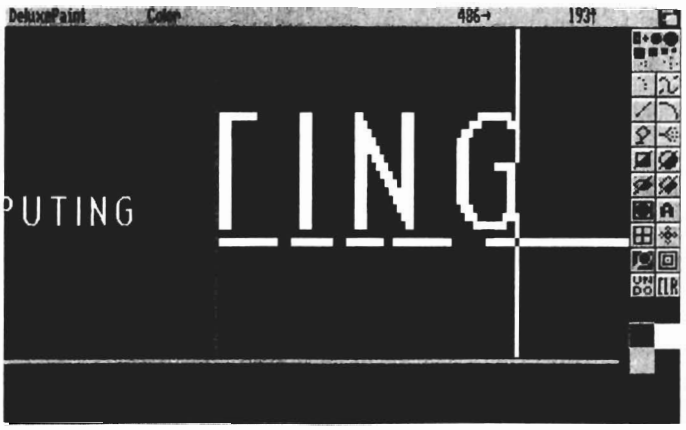
the icon's exact location on the picture. While in DeluxePaint, I activated the coordinates options (listed as Coords) under the far right-hand Prefs menu. The x and y coordinates of the graphics cursor is now visible along the right hand side of the menu bar. The x coordinate increases from zero on the left to 639 on the right. The y coordinate increases from zero to 199, *bottom* to *top*. I used the coordinates to define a box around the icon by the upper left and the lower right hand corners.

Unfortunately, AMOS Pro measures the y coordinate opposite from DeluxePaint. The AMOS Pro y coordinate increases from zero to 199, *top* to *bottom*. Once you have the upper left-hand corner coordinates (see Figure 4), you subtract the y coordinate from 199 to get the correct AMOS Pro value. You then subtract the lower right-hand value from 200. The 199 and 200 values are correct. They differ because the zero coordinate is significant when using the upper left value and it is not on the lower right value. You use the computed values in the Get Icon command. This AMOS command cuts the icons out of the DeluxePaint picture and stores them in a memory bank. See program Listing 2 for an example program, which I used to create the draw5.0.Abk file. The next problem was how to animate the Software Giveaway icon.

The Software Giveaway animation uses a BOB. I cut the BOB out immediately after the icon (see Listing 2, line 16). Icons 9 through 16 are blank icons. The program uses the blank icons to "erase" their respective icons one through eight. For example, icon 6 (The WINNER is ...) is "erased" by icon 13 by laying icon 13 over icon 6.

Once I had everything cut out, the program displayed each icon and then erased it. This was so I could tell if the icons were complete and not clipped. The last line of the program saves the information to a bank file named "draw5.0.Abk." You can use this file with any AMOS program.

If you save the memory bank with the program source code, you could delete line 7 of Listing 1. However, I could not get a compiled, standalone version of the program to run without crashing my computer. I use AMOS Pro Compiler, version 2.0. If I compiled the program without the memory bank included, it would run correctly. To erase the BOB and icon memory banks, select the AMOS Editor, Direct Mode and type the following:

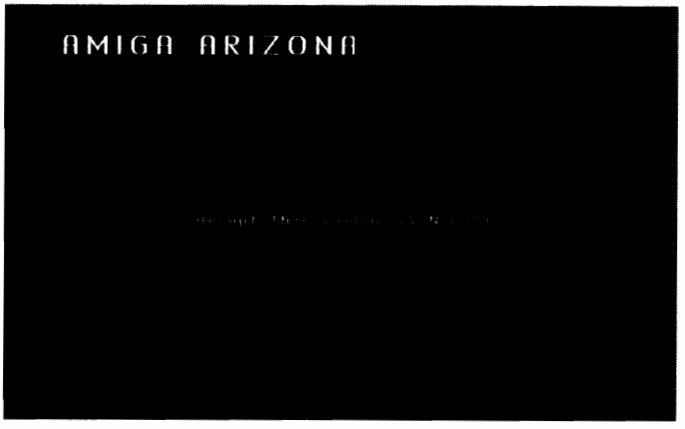



```
Erase 1<CR>, erases the BOB bank
Erase 2<CR>, erases the icon bank
```

If you compile the program, you should not include the memory banks in the saved AMOS program file.

An interested problem surfaced involving the Software Giveaway BOB. I noticed that when I cleared the program screen, a Software Giveaway icon remained on the screen. I later discovered it was my BOB and not an icon. Draw5.0 controls the animation through lines 41 to 49. Notice that line 48 places the icon over the BOB and then line 49 turns the BOB off. This action removes the BOB from the screen. If you would like to experiment with this, delete line 49 and run the program. When you exit on the first selection, you will notice that the BOB remains visible. Once I had fixed the BOB problem, I had one final screen display problem that affected the aesthetics of the program.

The program opens the working screen at line number three. Notice the screen size of 640 x 300. This is 50 percent larger than a 640 x 200 screen. This was necessary because with a normal screen display, I got a fragmented, one-pixel deep line across the top of the

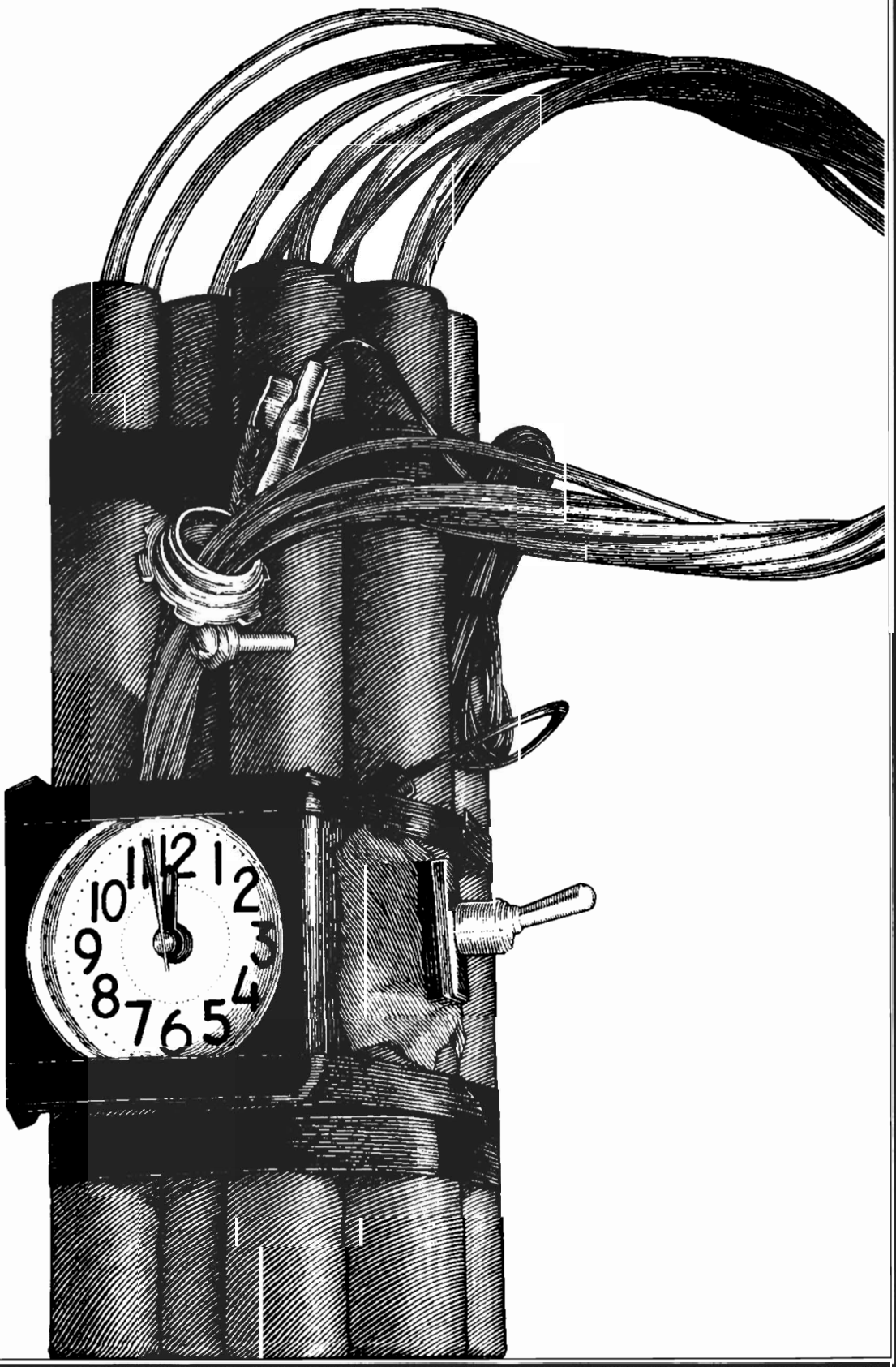


screen when the program returned to the selection process (lines 54 to 128). Eventually, I shifted the display up (reference line eight) to move the top of the screen out of view. I increased the screen size until the line no longer showed. If you would like to see the line, change line three from 300 to 200 and comment out line eight.

I have described a technique you can use to get the Amiga system date into an AMOS Professional program. I have given you the basis for a procedure that can read and write text files and make the information available for program use. Finally, I have showed one way to create and save a BOB and Icon bank, and I have pointed out a few lessons learned while writing this program. I hope you find this article beneficial and that you will pass along your AMOS Professional programming insight to the rest of us.

```
'
'                Draw5.0
' made especially for the AMAZ Users Group
'              Glendale, AZ
'
'              Copyrighted
'                  by
'          T. Darrel Westbrook
'
'
1 Set Buffer 64
2 Amos To Back
3 Screen Open 1,640,300,8,Hires
4 Wind Open 1,0,0,80,25
5 Flash Off
6 Hide : Rem turn off the mouse pointer
7 Load "draw5.0.Abk" : Rem load the icon bank into bank 2
8 Screen Display 1,130,25,640,300
9 Double Buffer
10 Curs Off

11 _MAX_ARRAY_SIZE=200
12 Dim LINE$(_MAX_ARRAY_SIZE) : Rem make it large enough
to hold members text file
13 Dim _ARRAY$(_MAX_ARRAY_SIZE,3)

14 Global LINE$(),_ARRAY$()
15 Global _SELECTION
16 Global FILENAME$
17 Global DIAMOND_FONT
18 '
19 ' to use within a string
20 Global C0$,C1$,C2$,C3$,C4$,C5$,C6$,C7$
21 '
22 Global DATE$,_DOW$,MONTH$,DAY$,MON$,YEAR$
23 Global _DEFAULT$,MEMFILE$,WINFILE$,LOSFILE$
24 Global WIN_FLAG,FILE_FLAG,LOS_FLAG

25 WIN_FLAG=False : Rem when True, then don't load date
in files
26 LOS_FLAG=False : Rem when True, then don't load date
in files

27 Proc _SET_COLORS : Rem set program colors
28 Palette $666,$F0,$610,$F0F,$FF0,$7,$F00,$0
29 Colour Back($666) : Rem make the background color the
same as the paper color
30 Paste Icon 184,87,1 : Rem - INITIALIZING -
31 Amos To Front

32 Proc _GET_FONTS : Rem find diamond 20 font
33 Proc _GET_SYSTEM_DATE : Rem get system date
34 Proc _GET_FILE_INFO : Rem get pathnames
35 FILENAME$=MEMFILE$ : Rem set to members.txt file
36 Proc _TEXT_FILE_READ : Rem load line$() array
37 LINE_SIZE=Param : Rem the returned length of the
_array$() array used
38 Set Font DIAMOND_FONT : Rem diamond 20 font is number
4 in ROM for OS 2.1
39 Set Text 4 : Rem italics


40 RESTART:
41 Cls 0 : Rem clear screen with color zero
42 For N=-28 To 38 : Rem start the BOB offscreen and
scroll down
43    Bob 1,45,N,1
44    Wait Vbl
45    Wait Vbl : Rem one Vbl wasn't slow enough
46 Next N
47 Paste Icon 45,20,4 : Rem AMIGA ARIZONA icon
48 Paste Icon 45,38,3 : Rem Software Giveaway over the
BOB
49 Bob Off 1 : Rem remove the bob from the screen
50 Randomize Timer
51 For N=1 To LINE_SIZE : Rem zero out arrray selection
52   _ARRAY$(N,2)=""
53 Next N


54 WINNER_SELECT:
55 Paste Icon 106,180,5 : Rem Press left mouse ...
56 Do
57    K=Mouse Click
58    If K Then Exit
59    Multi Wait : Rem give the rest of the system some
CPU time
60 Loop
61 _CONTINUE_FLAG=False
62 For N=1 To LINE_SIZE : Rem ensure there is someone
left that hasn't been picked yet
63    If _ARRAY$(N,2)=""
64       _CONTINUE_FLAG=True
65    End If
66 Next N
67 If Not _CONTINUE_FLAG : Rem _CONTINUE_FLAG is False
68    Cls 0
69    Paste Icon 45,95,8 : Rem All names have been ...
70    Clear Key : Rem clear keyboard buffer
71    Wait Key
72    Goto _EXIT
73 End If
74 Proc _RANDOM : Rem get a random selection, not picked
yet
75 Paste Icon 106,180,12 : Rem erase icon 5
76 Paste Icon 194,93,6 : Rem the WINNER ...
77 _LEN_OF_NAME=Int((640-
(12*Len(_ARRAY$(_SELECTION,1))))/2)
78 Ink 7,0
79 Text _LEN_OF_NAME,135,_ARRAY$(_SELECTION,1)
80 Gr Writing 1 : Rem replace any existing graphics with
new graphics, default condition
81 Ink 3,0
82 Gr Writing 0 : Rem only draw graphics using current
ink color
83 Text _LEN_OF_NAME+4,134,_ARRAY$(_SELECTION,1)
```

```
84 Gr Writing 1 : Rem replace any existing graphics with
new graphics, default condition
85 Paste Icon 175,150,7 : Rem Accept this winner ...
86 Curs Off
87 Clear Key
88 Do
89   AA$=Upper$(Input$(1)) : Rem get a single input &
make it upper case
90   If AA$="Y" or AA$="N" or AA$="E" or AA$="R"
91      Exit
92   End If
93 Loop
94 Paste Icon 194,93,13 : Rem erase icon 6, the WINNER is
...
95 Paste Icon 175,150,14 : Rem erase icon 7, Accept this
winner ....
96 Paste Icon 0,120,16 : Rem erase name
97 If AA$="Y" : Rem accept as the winner
98    '
99   ' update the winner file
100   '
101  FILENAME$=WINFILE$ : Rem assign winners.txt as
filename
102  FILE_FLAG=True : Rem FILE_FLAG is True for the
winner route
103  _ARRAY$(_SELECTION,2)="-1" : Rem put in don't use
again flag
104  Proc _TEXT_FILE_WRITE[True] : Rem append to end of
file
105  Goto WINNER_SELECT : Rem go get another winner
106 Else If AA$="N" : Rem don't accept as the winner
107   '
108   ' need to log in the looser file
109   '
110  FILENAME$=LOSFILE$ : Rem open up the losers.txt
111  FILE_FLAG=False : Rem FILE_Flag is False for the
loser route
112  _ARRAY$(_SELECTION,2)="-1" : Rem put in don't use
again flag
113  Proc _TEXT_FILE_WRITE[True] : Rem append to end of
file
114  Goto WINNER_SELECT : Rem got get another winner
115 Else If AA$="E" : Rem exit
116   Goto _EXIT
117 Else : Rem AA$ = "R", restart
118   Cls 0
119   Goto RESTART
120 End If
121 Goto WINNER_SELECT : Rem here as a catch all


122 _EXIT:
123 Cls 0
124 T$="assign "+Chr$(34)+"Draw:"+Chr$(34)
125 Exec T$ : Rem get rid of assignment of DRAW:
126 Wind Close
127 Screen Close 1
128 End : Rem end of Draw5.0 program

129 Procedure _RANDOM
130   Rem generate a random selection
131   Shared LINE_SIZE
132   Rem based on the number of members present
133   Do : Rem until a new name is selected
134     Do : Rem until _SELECTION is not zero
135       _SELECTION=Int(Rnd(LINE_SIZE))
136       If _SELECTION<>0
137          Exit
138       End If
139     Loop
140    If _ARRAY$(_SELECTION,2)<>"-1" : Rem if -1 then
selected before
141       Exit
142     End If
143   Loop
144 End Proc : Rem end of '_RANDOM'

145 Procedure _TEXT_FILE_READ
146   Rem
147   Rem must dimension line$() prior to calling the
procedure
148   Rem
149  Shared FILENAME$,LINE$()
150  COUNT=0 : Rem keep track of _ARRAY$() Length
151  Open Random 1,FILENAME$ : Rem a text file all at
once
152  SIZE=Lof(1)
153  Field 1,SIZE As A$
154  Get 1,1
155  Close 1
156  A=0 : X$="" : X=0
157  Do : Rem determine the end of a$ character
158    If Right$(A$,1)<>Chr$(32)
159       A$=Mid$(A$,1,SIZE-X)
160       SIZE=SIZE-X
161       Exit
162     End If
163     Inc X
164  Loop
165  If Right$(A$,1)<>Chr$(10) : Rem find the line
feeds (LF, decimal 10)
166     A$=A$+Chr$(10) : Rem        and count them,
gives the total
167      Inc SIZE : Rem             number of lines in
the file
168   End If
169   For N=1 To SIZE : Rem find the places where the
LFs are
170    X=Instr(A$,Chr$(10),N)
171     N=X+1 : Rem step pass the LF
172
X$=X$+Right$("0000"+Right$(Str$(X),Len(Str$(X))-1),4) :
Rem remember the position of the LF
173   Next N
174  A=Len(X$)/4 : Rem the number of lines of text
175   Y=1 : B=0
176   For N=1 To A : Rem separate the file into lines
177    X=Val(Mid$(X$,N*4-3,4))
178     Inc B : Rem line counter for the text file
179    LINE$(B)=Mid$(A$,Y,X-Y)
180     Proc TRIM[LINE$(B)] : Rem remove leading and
trailing blanks
```


12
1
2
3
4
5
6
7
8
9
10
11

Don't you think it's time you got *Amazing Computing*?

Amazing Computing for the Commodore Amiga, AC's GUIDE and AC's TECH provide you with the most comprehensive coverage of the Amiga. Coverage you would expect from the longest running monthly Amiga publication.

The pages of Amazing Computing bring you insights into the world of the Commodore Amiga. You'll find comprehensive reviews of Amiga products, complete coverage of all the major Amiga trade shows, and hints, tips, and tutorials on a variety of Amiga subjects such as desktop publishing, video, programming, & hardware. You'll also find a listing of the latest Fred Fish disks, monthly columns on using the CLI and working with ARexx, and you can keep up to date with new releases in New Products and other neat stuff.

AC's GUIDE to the Commodore Amiga is an indispensable catalog of all the hardware, software, public domain collection, services and information available for the Amiga. This amazing book lists over 3500 products and is updated every six months!

AC's TECH for the Commodore Amiga provides the Amiga user with valuable insights into the inner workings of the Amiga. In-depth articles on programming and hardware enhancement are designed to help the user gain the knowledge he needs to get the most out of his machine.

For subscription information, call

1-800-345-3360

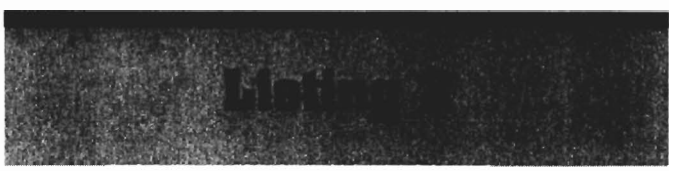
```
181      LINE$(B)=Param$
182      If Len(LINE$(B))<>0 : Rem don't load any blank
lines
183        Inc COUNT : Rem the membership array
counter, may be different from SIZE
184       _ARRAY$(COUNT,0)=Mid$(LINE$(B),1,3) : Rem
get membership number
185       Z$=Mid$(LINE$(B),4,Len(LINE$(B))) : Rem get
members name
186         Proc TRIM[Z$]
187        _ARRAY$(COUNT,1)=Param$
188       End If
189      Y=X+1 : Rem get past the LF
190    Next N
191 End Proc[COUNT] : Rem end of '_TEXT_FILE_READ' return
the size of the array

192 Procedure _TEXT_FILE_WRITE[_ADD_FLAG]
193
194    ' write a text file to disk one line at a time
195    ' assume text file FILENAME$ is the file to write
to
196    ' and array$() is the array passed to the proce-
dure to write to disk
197    ' and _ADD_Flag if True will append to file,
otherwise it erases and starts again
198     '
199    ' FILE_FLAG along with WIN_FLAG & LOS_FLAG
ensures that the
200     ' date is loaded in the file only once
201     '
202    If _ADD_FLAG
203      If Exist(FILENAME$)
204        Open Random 1,FILENAME$
205         Field 1,1 As A$
206        Pof(1)=Lof(1) : Rem get len of file and set
file pointer there
207         RECORD_NUM=Pof(1)
208       Else
209         Rem file does not exist
210         Open Random 1,FILENAME$
211         Field 1,1 As A$
212        Pof(1)=1 : Rem set file pointer to first
record
213         RECORD_NUM=0
214       End If
215    Else : Rem start a new file
216       Rem erase file
217      If Exist(FILENAME$)
218         Kill FILENAME$
219       End If
220      Open Random 1,FILENAME$
221       Field 1,1 As A$
222      RECORD_NUM=0
223     End If
224   If FILE_FLAG=True and WIN_FLAG=False : Rem load
date in winner file
225      _LEN_OF_STR=Len(DATE$)
226       For J=1 To _LEN_OF_STR
227         A$=Mid$(DATE$,J,1)
228          Inc RECORD_NUM
229          Put 1,RECORD_NUM
230        Next J
231       A$=Chr$(10) : Rem put a line feed on the end of
the string
232       Inc RECORD_NUM : Rem Increment one for the line
feed
233       Put 1,RECORD_NUM
234      WIN_FLAG=True : Rem don't put in any more dates
this run
235    Else If FILE_FLAG=False and LOS_FLAG=False : Rem
load date in loser file
236      _LEN_OF_STR=Len(DATE$)
237       For J=1 To _LEN_OF_STR
238         A$=Mid$(DATE$,J,1)
239          Inc RECORD_NUM
240          Put 1,RECORD_NUM
241        Next J
242       A$=Chr$(10) : Rem put a line feed on the end of
the string
243       Inc RECORD_NUM : Rem Increment one for the line
feed
244       Put 1,RECORD_NUM
245       LOS_FLAG=True : Rem don't put in any more dates
this run
246    End If : Rem end of date loading conditional

247   _LOAD_STRING$=_ARRAY$(_SELECTION,0)+"
"+_ARRAY$(_SELECTION,1)
248   _LEN_OF_STR=Len(_LOAD_STRING$)

249    For J=1 To _LEN_OF_STR
250      A$=Mid$(_LOAD_STRING$,J,1)
251       Inc RECORD_NUM
252       Put 1,RECORD_NUM
253     Next J
254    A$=Chr$(10) : Rem put a line feed on the end of
the string
255    Inc RECORD_NUM : Rem Increment one for the line
feed
256    Put 1,RECORD_NUM : Rem load the data into the file
257     Close 1
258 End Proc : Rem end of '_TEXT_FILE_WRITE'

259 Procedure LINE_2_ARRAY[_MAX_SIZE]
260    For N=1 To _MAX_SIZE : Rem load the _ARRAY$()
261      _ARRAY$(N,1)=LINE$(N)
262     Next N
263 End Proc : Rem end of 'LINE_2_ARRAY'

264 Procedure TRIM[A$]
265    For N=1 To Len(A$)
266      If Mid$(A$,N,1)<>" " Then A$=Mid$(A$,N,Len(A$)-
N+1) : Exit
267     Next N
268    For N=Len(A$) To 1 Step -1
269      If Mid$(A$,N,1)<>" " Then A$=Mid$(A$,1,N) :
Exit
270     Next N
271 End Proc[A$]

272 Procedure _GET_FONTS
```

```
273    '
274    ' diamond 20 font, should be ROM font number 4
275    ' loading this so the program can access the font
276    '
277    FLAG=0
278    Get Fonts : Rem load all fonts in an array Font$
279    N=0
280    Do
281       Inc N
282      Trap N$=Font$(N) : Rem ensure we don't go past
end of array
283       If Errtrap
284          Exit
285       End If
286    If Lower$(Mid$(Font$(N),1,12))="diamond.font"
and Mid$(Font$(N),31,2)="20"
287         DIAMOND_FONT=N : Rem diamond 20 font found
288           Exit
289       End If
290    Loop
291   If DIAMOND_FONT=0 : Rem occurs is diamond 20 not
found
292       Clw : Rem      this is a fatal error and the
program
293       Curs Off : Rem will not run
294       Pen 0 : Rem write with color zero
295       Locate 1,8
296      Centre "I can not locate the "+C4$+"diamond
20"+C0$+" font I need to operate this program."
297       Locate 1,10
298      Centre "Check the "+C4$+"FONTS: "+C0$+"path to
ensure "+C4$+"diamond 20"
299        Pen 0
300       Locate 1,12
301      Centre "is available.  Rerun program after the
fonts are available."
302       Paste Icon X,120,2 : Rem - Press any key ...
303       Clear Key
304       Wait Key
305       Clw
306       Window 2
307       Wind Close
308       Screen Close 2
309      FLAG=-1 : Rem exit out of the program
310    End If : Rem end of _'GET_FONTS'
311 End Proc[FLAG] : Rem end of '_GET_FONTS'

312 Procedure _GET_SYSTEM_DATE
313    Exec "date > RAM:Date_Data.txt" : Rem use AmigaDOS
to get the current system date
314   Open Random 1,"RAM:Date_Data.txt"
315    SIZE=Lof(1)
316    Field 1,SIZE As A$
317    Get 1,1
318    Close 1
319   Kill "RAM:Date_Data.txt"
320   X=Instr(A$,Chr$(32),N) : Rem get day of the week
321   _DOW$=Mid$(A$,1,X-1)
322   A$=Mid$(A$,X+1,Len(A$)) : Rem strip off day of the
week
323   X=Instr(A$,Chr$(32),N) : Rem get date
```



```
324   DATE$=Mid$(A$,1,X-1) : Rem format of dd-mmm-yy
325    Proc TRIM[DATE$]
326    DATE$=Param$

327    X=Instr(DATE$,Chr$(45)) : Rem get day
328   DAY$=Mid$(DATE$,1,2)
329   X$=Mid$(DATE$,X+1,Len(DATE$)) : Rem strip off day

330    X=Instr(X$,Chr$(45),N) : Rem get month
331   MONTH$=Mid$(DATE$,4,3)
332   YEAR$=Mid$(DATE$,8,2)
333   If Val(YEAR$)>50 Then YEAR$="19"+YEAR$ Else
YEAR$="20"+YEAR$
334   Y$="JanFebMarAprMayJunJulAugSepOctNovDec"
335    For N=1 To 12
336      If MONTH$=Mid$(Y$,N*3-2,3)
337
MON$=Right$("00"+Right$(Str$(N),Len(Str$(N))-1),2)
338          Exit
339       End If
340    Next N
341   DATE$=MON$+"-"+DAY$+"-"+YEAR$
342 End Proc : Rem end of '_GET_SYSTEM_DATE'

343 Procedure _SET_COLORS
344    Screen 1
345     '   0     1     2       3       4      5     6
7
346    ' gray, green, brown, magenta, yellow, blue, red,
black
```

```
347    ' $666  $0F0   $610    $F0F    $FF0   $007  $F00
$0
348  Default Palette $666,$F0,$610,$F0F,$FF0,$7,$F00,$0
349  Palette $666,$F0,$610,$F0F,$FF0,$7,$F00,$0
350   C0$=Pen$(0) : Rem gray
351   C1$=Pen$(1) : Rem green
352   C2$=Pen$(2) : Rem brown
353   C3$=Pen$(3) : Rem magenta
354   C4$=Pen$(4) : Rem yellow
355   C5$=Pen$(5) : Rem blue
356   C6$=Pen$(6) : Rem red
357   C7$=Pen$(7) : Rem black
358   Colour Back($666)
359   Cls 0 : Rem use this to clear screen in gray color
360   Curs Off : Rem I don't wand to see the cursor
361 End Proc : Rem end of '_SET_COLORS'

362 Procedure _GET_FILE_INFO
363    '
364    ' Set up pathnames, assume files Members.txt,
Winners.txt
365    ' and Losers.txt are located in the Draw:Draw/
directory
366    '
367    Do
368      If Not Exist("Draw:") : Rem checking for the
Draw: assign
369        If Not Exist("Members.txt")
370           Show On : Rem members.txt is not on
current directory
371          T$=Fsel$((Dir$),"*.*","Please select a
directory to assign 'Draw:' to.","")
372           _DEFAULT$=Dir$ : Rem assumes that the
losers.txt and winners.txt files on in the same directory
373            Hide : Rem turn mouse off
374           Trap Assign "Draw:" To _DEFAULT$
375         Else : Rem Members.txt found
376           _DEFAULT$=Dir$
377           Trap Assign "Draw:" To _DEFAULT$
378         End If
379       Else
380          Exit
381       End If
382    Loop
383  MEMFILE$=_DEFAULT$+"Members.txt" : Rem assign all
files to same directory
384  WINFILE$=_DEFAULT$+"Winners.txt"
385  LOSFILE$=_DEFAULT$+"Losers.txt"
386    Do
387      If Not Exist(MEMFILE$)
388         Show On : Rem turn mouse on
389       T1$=Fsel$((Dir$),"*.*","Please select the
Members file","")
390        T1$=Dir$ : Rem set directory to the one just
selected
391         Hide : Rem turn mouse off
392        MEMFILE$=T1$+"Members.txt" : Rem reset
members file pathname
393        WINFILE$=T1$+"Winners.txt" : Rem assume that
this is the same
394        LOSFILE$=T1$+"Losers.txt" : Rem  location
for the other files
395       Else
396          Exit
397       End If
398    Loop
399    Do
400      If Not Exist(WINFILE$)
401         Show On : Rem turn mouse on
402       T2$=Fsel$((Dir$),"*.*","Please select the
winners file","")
403        T2$=Dir$ : Rem set directory to the one just
selected
404         Hide : Rem turn mouse off
405        WINFILE$=T2$+"Winners.txt" : Rem reset
winners file pathname
406        LOSFILE$=T2$+"Losers.txt" : Rem assume
LOSERS.TXT is in the same directory
407       Else
408          Exit
409       End If
410    Loop
411    Do
412      If Not Exist(LOSFILE$)
413         Show On : Rem turn mouse on
414       T3$=Fsel$((Dir$),"*.*","Please select the
Losers file","")
415        T3$=Dir$ : Rem set directory to the one just
selected
416         Hide : Rem turn mouse off
417        LOSFILE$=T3$+"Losers.txt" : Rem reset the
location of losers.txt file
418       Else
419          Exit
420       End If
421    Loop
422 End Proc : Rem end of '_GET_FILE_INFO'
```



```
'
' Program, Draw5_0_Get_Pic.AMOS
'
' Utility Program to cut AMOS icons out of
' an IFF picture for use in the Draw5.0
' program.
'
'         Coipyrighted
'               by
'     T. Darrel Westbrook
'
_DIR$="Draw5_0_Parts.pic"
Screen Open 0,640,400,8,Hires
Erase 1
```

```
Erase 2
Flash Off
Load Iff _DIR$

' 0     1     2       3        4      5     6    7
' gray, green, blue, magenta, yellow, green, red, black
Default Palette $666,$F0,$610,$F0F,$FF0,$7,$F00,$0
Palette $666,$F0,$610,$F0F,$FF0,$7,$F00,$0
Colour Back($666)
Paper 0


Get Icon 1,315,23 To 586,38 : Rem - initializing -
Get Icon 2,7,72 To 297,81 : Rem - Press ...
Get Icon 3,12,44 To 226,60 : Rem 'software giveaway'
Get Bob 1,12,44 To 226,60 : Rem get BOB 'Software
Giveaway'
Get Icon 4,12,23 To 292,38 : Rem get the amiga arizona
Get Icon 5,12,89 To 438,97 : Rem Press left mouse ...
Get Icon 6,26,123 To 277,136 : Rem the WINNER ...
Get Icon 7,273,50 To 561,58 : Rem Accept this winner ....
Get Icon 8,13,106 To 562,115 : Rem All the names ....
'
' make backgound erase squares to lay over the above
icons
' when I want to 'erase' them
Get Icon 9,0,141 To 281,166 : Rem used to erase icon 1 &
4
Get Icon 10,0,141 To 291,150 : Rem erase icon 2
Get Icon 11,0,141 To 214,178 : Rem used to erase icon 3
Get Icon 12,0,141 To 427,149 : Rem erase icon 5
Get Icon 13,0,141 To 252,154 : Rem erase icon 6
Get Icon 14,0,141 To 289,149 : Rem erase icon 7
Get Icon 15,0,141 To 550,150 : Rem erase icon 8
Get Icon 16,0,141 To 640,168 : Rem erase name selected
Get Icon Palette

'
' look at each icon
'
Clw
Paste Icon 10,100,1
Wait Key
Paste Icon 10,100,9
Wait Key

Paste Icon 10,100,2
Wait Key
Paste Icon 10,100,10
Wait Key

Paste Icon 10,100,3
Wait Key
Paste Icon 10,100,11
Wait Key

Paste Icon 10,100,4
Wait Key
Paste Icon 10,100,9
Wait Key

Paste Icon 10,100,5
Wait Key
Paste Icon 10,100,12
Wait Key

Paste Icon 10,100,6
Wait Key
Paste Icon 10,100,13
Wait Key

Paste Icon 10,100,7
Wait Key
Paste Icon 10,100,14
Wait Key

Paste Icon 10,100,8
Wait Key
Paste Icon 10,100,15
Wait Key
'
'
Save "draw5.0.Abk" : Rem save in a bank file
'
'
End
```

✔

The complete set of listings can be found on the AC's TECH disk.

Please write to:
T. Darrel Westbrook
c/o AC's TECH
P.O. Box 2140
Fall River, MA 02722

Writing a Function Genie for Professional Draw

Like most major Amiga software developers, Gold Disk has provided ARexx functionality for their primary products. Users of *Professional Page* and *Professional Draw* (PPage and PDraw), as well as *Professional Calc*, have this capability. However, I suspect that many users do not tap the power provided because the documentation is so poor: it's just plain wrong in places. This article focuses on a full-featured application for PDraw: an ARexx program (called a Genie by Gold Disk) that can draw calendars for any "reasonable" year or month. I have done much hair-pulling—what does that fourth argument really do?—and help-line phoning to supplement the documentation; I hope to save some readers some time and inspire them to write that Genie that has been living only in the back of their minds.

The ARexx interface that Gold Disk provides in both PDraw and PPage, starting at version 3.0 for both products, is a function host rather than the more common command host. What this means in a practical sense is that you don't have to keep switching to a host with the rexx *address* statement to talk to it; the functions seem to be a part of ARexx itself. A side effect of the implementation is that all functions can be called by their documented name from Function Genies that are running from within PDraw, but if an external ARexx program is drawing something, then the function names have to be prefixed with `pdm_`. I found that it was better to include the prefix all the time, since it does not cause a problem when it appears in a Genie, and you never know when you might export a chunk of code to an external rexx program. In this article, I've dropped the prefix, but I've retained Gold Disk's method of using capital letters as the first letter of each word in the function names. Another side effect is that program file names have to end in

Creating a Calendar Beginning October 1582

by Keith D. Brown

.pdrx if they are to be seen as Genies. These files have to be in the rexx: directory with all your other rexx programs, to be noticed by PDraw.

Gregorian vs. Julian

It seems that I either have too many calendars given to me at Christmas, or none. Last year must have been of the second case, since I started the program then. I've recently tidied it up, made it work from the CLI and extended the range of years that it covers. There is a limitation of the Arexx function date(): it does not work for dates earlier than January 1, 1978, nor later than January 1, 2100. A surprising number of people, when shown a program that generates any calendar, want to see their birth month, and since most of the people I know are older than 15, this became a challenge. The calendar that we use now is the Gregorian: it was instituted in October of 1582. That month, if we had to live through it now, would cause money lenders great grief: it was 10 days shorter than usual! This was the amount of error that had accumulated in the previous Julian calendar. The Gregorian calendar better accounts for the fact that a year is 365.242199 days; its system of leap years averages the year's length to be 365.2425 days. The remaining difference will accumulate in about 4000 years such that a correction of -1 day would then be needed. I don't know if this additional correction is built in to the official calendar, or if it is, when—perhaps the year 5584 won't be a leap year?—or if anybody cares, so I ignored it. The program will draw any calendar, month or year, for any year greater than 1582. It does not attempt the very different calendar for 1582, nor any year earlier. For historical purposes, it should be noted that not all countries adopted the Gregorian calendar at the same time; if one is researching dates, one has to know the calendar in effect in the countries of interest at the time.

The Calendar Program.

Now to the program. Since PDraw is being used to draw the calendars, the user can take the product of the Genie and embellish it in any way desired: colors can be changed, artwork added to make your own customized Christmas gifts, assuming your printer is up to high-resolution color, months can be cut from two years to make a personalized school year calendar, and so on. Two potential features that I didn't tackle are holidays and phases of the moon. To implement holidays, I would use an external file containing dates (absolute: 25 Dec=Christmas and relative: Thu 4 Nov=US Thanksgiving). The problem here is that religious holidays are not always consistent with the Gregorian calendar. I looked at the Virtual Reality Laboratories program *Distant Suns* to see about the phases of the moon, but concluded that I'd have to do some trig to determine the phase of the moon; this is available within the program but not directly through the ARexx port, and the program is already slow enough. I'll leave these two features as exercises for the Amiga community.

To use the program from the CLI type: `rx DrawCalendar.pdrx arg` where *arg* is either a year or of the form mm/yyyy. The presence of the slash is used to decide whether a single month or a whole year calendar is desired. To use as a Genie, click on the Genie tool near the top right of the PDraw screen. Double click on the DrawCalendar Genie name in the list, and the program will ask you for your choices. Defaults are set up to be the current month and year. It's safest to ensure beforehand that there is enough room on your page to receive the whole calendar. If it doesn't fit, then some of the text can be lost at the break between the page and the "Art Board" if the sizing tool is used to scale it down. (If you get into this situation, modify the page size to fit the calendar, not the opposite. Gold Disk seem to have implemented a conditional Black Hole at the edge of the page.) It takes about seven minutes on my A3000 to draw a year's calendar. A month takes about half a minute.

The structure of the program (see listing) is straightforward. The first executable statement (the parse) is there to get an argument from the CLI if the program was started that way. After a few initializations, a block of assignments allow the user to customize the performance of the program. Such factors as sizes, the font, line widths, colors and various operational considerations are user-configurable by editing these well documented lines.

Two text heights are controlled by the user-configurable variables day_name_height and month_name_height, which are defined as proportional to the week height, which in turn is one sixth of the variable month_height. The position of the day names is fixed in the program (in the routine DrawMonth) as .05" above the starting point. The position of the month name, however, is proportional to the height of the day names: this variable can be changed since the best position will be a function of the font used. This is controlled by variable day_month_space. It is defined in units of day_name_height so that a value of 1.3 would cause position 4 in the figure to be at 130% of the height of the character as defined by day_name_height. It is up to the user to allow for enough room between rows of months in the year calendars; the value of the variable year_row_sep has to be large enough to allow these two rows of text. This method of controlling options was chosen, rather than a GUI interface, to keep it simple. PDraw does provide some means to interact with the operator through requesters, but doesn't have a fully generic system (nor should it—there are enough inventions of that wheel out there already) that would be efficient enough to implement a Preferences-type screen. To edit the program, you had better use the Gold Disk provided Genie editor, even if you don't like it, to make changes to a Genie, since changes made by an external editor will not be apparent to PDraw unless it is restarted.

The code following the configuration area checks the argument to decode the user's choice. Note that if no choice is given, or if this program is executed from within PDraw as a Genie, then a requester will pop up later to request the year and month input. This can't be done right away since at this point it is not known whether PDraw is running. That is the next step. The "if ~show('P','PDRAWAREXX')" statement is an easy way to find out if PDraw is executing. An aside here—Gold Disk does not allow more than one copy of PDraw to access ARexx. There is only one port name possible, and the first

copy of PDraw gets it: this doesn't conform to the Commodore User Interface Style Guide, which would have the ports named PDRAWAREXX.1, etc. The same problem exists with version 3.0 of PPage; I haven't seen version 4.0. This means that if PDraw is started while another copy is running, your calendar will go to the first copy regardless of what you want. If you aren't sure if a PDraw copy is talking to ARexx, just click on the Function Genie tool: if the Genie list box appears then that copy is ARexx-aware.

The next blocks of code interact with the user to get the desired calendar form and dates (if the argument is supplied on the CLI, then the calls to SelectFromList and GetForm are skipped) and adjust the size parameters based on the user-changeable variables. The DrawMonth subroutine is called once for the single-month calendar case and 12 times for the year, adjusting the starting positions of each month as it goes along, in either a 3 x 4 or 4 x 3 pattern. Following this, the value of the user-changeable variable print_after_drawing is used to decide which action to take. The last action designed into the main program turned out to be not practical: the PDQuit was called right after the print call; it is not possible in version 3.0 of PDraw to wait for the dot matrix print command to finish. The obvious workaround, to add a delay, did not work either; it seems that Gold Disk's Pause(seconds) function does not pause! The dot matrix print function is a good demonstration of the documentation quality: there is no mention of any dot matrix printing functions in the version 3.0 supplement manual, but the functions exist. I inferred their presence by similarity to the PPage documentation, and then found their arguments by trial and error. This led me to conclude that the third argument, *sync*, as used in PPage, is not implemented in PDraw; this is what is needed to wait for the function to complete. I've commented out this section since I couldn't get the PDQuit to not be PDQ!

Functions and Procedures

The DrawMonth routine is the top level of a hierarchy of routines: it is responsible for all graphics drawn to represent a month. It calls DrawDays to draw from one to seven days, and does this either four (a non leap-year February that starts on the first day of the week), five or six times.

The routine FirstDOY() is where the bypassing of the ARexx date() limitations begins. It uses the facts that a normal year has a whole number of weeks plus 1 day in it, and a leap year is one day longer. It starts at the known year 1978, which started on a Sunday, and counts either backwards or forwards to the desired year. It calls the function Leap(), which uses the Gregorian calendar's rules to determine leap years. Once the start-day is known, then the function FindAliasYear() is called to return a year that is in the accepted range. There are only 14 unique calendars, so it is an easy task. The years used in FindAliasYear() were chosen with the aid of another ARexx program. That program called date() to determine the first days and leap year status for years starting at 1978.

The two subroutines InitDayNames() and InitMonthNames() are simple. They both have some options, however. InitDayNames() can produce either one-character or three-character names, and they can start on Sunday, the North-American English norm, or Monday, as used in some other countries. InitMonthNames() can produce full names or three-character abbreviations. Both routines can produce upper-case strings if desired. One needs only to edit these two routines to produce calendars in other languages.

The LRCenter() procedure is needed to get around some of the awkwardness of PDraw. The way that PDraw handles text is quite different than one might think. There is quite a disparity between how text is handled on-screen with the mouse, and the set of functions and their arguments and returned values. As text is generated, at least one object id number is generated for each letter; however, a program cannot get these numbers directly. Even when converting text to graphics with the TextToGraphic() function, nothing is returned. The ruse that I used was to create dummy objects both before and after creating text that I wanted to control. The text could then be manipulated, in the case of LRCenter—grouped so that its center could be found and then changed, as if it were a regular object.

Quirks of Calls and Documentation Problems

•I could not get the Text() function to work. It is evident that the arguments are in a different order (*string,x,y*) than documented (*x,y,string*); however, that is not all. I could get it to draw only garbage characters. The savior here was that a search of all of the supplied Genies turned up an undocumented function that does work: FlowText(string,X,Y,Xmax,Ymax). FlowText will supply new-line action if Xmax is exceeded so multiple lines of text can be produced; however, if Xmax is large enough, then one line of text is generated. The fourth argument limits the number of lines produced; strangely a zero doesn't: it may be that a Ymax less than Y disables this limiting.

•If a program calls FlowText(), without calling InitText() first, then nothing happens—no text and no error message.

•For the most part, the SetXxx() and GetXxx() functions are matched, however there is no SetCenter() to match GetCenter().

•The CenterObj([obj],dir) function doesn't do what I wanted so I had to write my own function (LRCenter—already discussed) to center text with respect to fixed coordinates. The action produced by CenterObj is based on the positions and the widths of the objects selected. For two objects with center x coordinates x1 and x2 and widths w1 and w2, the resulting x coordinate is (x1+x2)/2 + (w2-w1)/4 if either a 1 or a 2 is supplied as the *dir* argument. Similar action, based on the vertical position and heights, is produced for the y coordinates for a *dir* of 0 or 2. This function is sometimes dangerous since it moves all objects that it is centering. It seems to me that a more useful function would be one that centers the chosen objects with respect to one object or a set of fixed objects. It'll be a simple matter to generalize LRCenter() to provide this function as a Genie. Note that PDraw coordinates refer to bounding boxes: center coordinates are the center of that box, not the center of area.

•GetUserText() only uses the first 20 characters of the *prompts* argument. I avoid this function: GetForm() can do anything that GetUserText() can do, and with more control and fewer limitations.

•The requesters produced by GetUserText(), SelectFromList() and GetForm() *can* be dragged, but the requester put up by Inform() is frozen in place.

•DrawGrid() cannot produce grids with horizontal lines. It has six arguments, not the four documented; the fifth is the number of vertical lines, but the sixth doesn't seem to do anything.

•The third argument to InitText() is the aspect ratio of the characters; this is not clear from the documentation.

•The *width* and *height* arguments to DrawEllipse() are radii; the documentation claims they are diameters.

•There are differences in the way that an external ARexx function controlling PDraw works, as compared to a Genie. A Genie has full control over the screen, and when it is running, a little clock shows the activity on the title bar; the program can refresh the screen after each action, or wait until completion. In an external program, however, the screen updates after each command, whether you want it to or not. As well, one can simultaneously use the mouse to select actions when an external script is running—often with strange results.

Conclusion

Having been forced to use *CorelDRAW* on a PC at work, I found that it had a lot of features built in to it with MBs of data and fonts, but in the home office I found that PDraw was no slouch. The capability to extend it by programming in ARexx, and call up these programs via the Function Genie tool, proved to be very powerful. This programmability provides superior extensibility compared to any structured drawing program that I know about on the PC.

Listing

```
/*
This Genie will draw either a month or a year's calendar.
Sizes and other options may be changed by editing values
of variables in the first block of code. If doing this it
is advised that the results be saved as another file e.g.
DrawCalendarMod.pdrx.
*/
/*
 * Usage: call as a Professional Draw Genie
 *  or CLI as  rx DrawCalendar.pdrx [mm/year|year]
 *
 *Copyright 1993 Keith D. Brown. Weston, Ontario, Canada.
 *
 *
 * Revision history:
 *      7-Oct-1993 K.D.Brown Release 1.0
 */
parse arg args
signal on error; signal on syntax; signal on break_c

/*********************************************************/
/* User-changeable variables                             */
/*  (i.e. mess with anything else at own risk)           */
/* Dimensions are in inches unless otherwise specified */
/*********************************************************/
/* for the general case: */
/*NEXT LINE change disk:directory to match your system */
start_PDraw_string = 'run work:PDraw/PDraw'
monday_start = 0        /* 0 = Sunday start,           */
                        /* 1 = Monday start            */
font = "GarthGraphic"   /* use this font for all text  */
text_thickness = 0      /* just use fill for characters*/
                        /* i.e. no outline             */
main_color = 'Black'     /* colour of the boxes        */
text_color = 'Black'     /* you guessed it!            */
full_month_names = 1    /* 0 for 3-letter month names, */
                        /* 1 for full names            */
right_justify_numbers = 1 /* puts a blank before single*/
                       /* digit numbers, 0 does not.  */
center_month_names = 1  /* 0 left-justifies, 1 centers */
center_day_names = 1    /* 0 left-justifies, 1 centers */
upper_month_names = 0    /* 1 upper case, 0 lower      */
upper_day_names = 0      /* 1 upper case, 0 lower      */
day_month_space = 1.6   /* times day_name_height, use a*/
                       /* smaller number if upper case*/
print_after_drawing = 2 /* 1 causes print_string to    */
                        /* execute after the calendar  */
                        /* has been drawn; 0 does not; */
                        /* 2 asks user whether to print*/
                         /* or not. */
print_string = 'succ = pdm_PrintPageDM(1,1)'/* none of */
                        /* the Print#?DM (DotMatrix)   */
                        /* commands are documented, but*/
                        /* PrintPageDM and PrintDocDM  */
                         /* exist and work.            */
/* I have neither a postscript printer, or the updated
   postscript version of PDraw so havn't tested this:
  Swap for line above to use postscript...
print_string = 'succ = pdm_PrintPagePS(1,1)'/* none of */
*/

/* for the one-month case: */
month_line_thickness = 2/* units are points           */
month_name_height = .90 /* times month_height         */
day_name_height =   .35 /* times month_height         */
month_width  = 6        /* exactly                    */
month_height = 4       /*6 weeks, doesn't include text*/
month_start_x = 0.5     /* top left coordinates       */
month_start_y = 1.6
month_char_pos = 0      /* 0=top-left, 1=centered;    */
                       /* size and pos'n of the day   */
                       /* numbers in the boxes.       */
short_day_name = 0     /* 0:'Mon'...'Sun' 1:'M'...'S' */

/* for the year case: */
year_line_thickness = 1 /* thin for smaller months    */
year_title_size = .666  /* height of year heading     */
year_month_name_height = 1.2  /* times month height   */
year_day_name_height =   .5   /* times month_height   */
year_month_width = 2    /* basic size of each month   */
year_3by4 = 0           /* 3 across, 4 rows: 0 is 4x3 */
year_month_height = 1.5 /* not counting text          */
year_column_sep = .5    /* space between months       */
year_row_sep = .75
year_start_x = .5       /*top left of the year calendar*/
year_start_y = 1.5
year_char_pos = 1       /* 0=top-left, 1=centered     */
year_short_day_name = 1 /* 'M'...'S'                  */
/*********************************************************/
/* End of user-changeable variables.                     */
/*********************************************************/

cr  = '0a'x
restart:  /* can get back here if user clicks RESTART  */
         /* on certain error message boxes */

PD_open = 0       /* assume that PDraw is not running  */

Started_PDraw = 0 /* use this to decide whether to exit*/
                 /* not used since exit can't wait for*/
                  /* print to end. */

    /* Decide if an argument has been supplied */
    /* (only matters if CLI-called):           */
type = ""
arg_supplied = ~(length(args)=0)
if arg_supplied then
  do
    if index(args,'/') = 0 then
      do
        type = "Year"
        year = args
      end
    else
      do
        type = "One Month"
        month = args  /* will parse later */
      end
  end

/* find out if PDraw is running; if not start it: */
if ~show('P','PDRAWAREXX') then  /* see if port exists */
  do
    address command   /* talk to dos for a short while */
    start_PDraw_string
    address         /* back to rexx to wait for PDraw  */
     do while ~show('P','PDRAWAREXX')/* This is brute */
        nop                          /* force, Is there */
      end                            /* a better way?   */
    PD_open = 1
    Started_PDraw = 1
  end
else
  do
```

```
    PD_open = 1 /* it is running already */
    call pdm_PDrawToFront()
  end

msg = PDSetup.rexx(2,0) /* Supplied by Gold Disk */
units = getclip(pds_units)
if msg ~= 1 then exit_msg(msg)

call pdm_UnselectObj() /* allow no other objects to be
                          affected by this Genie */
call init_month_names(full_month_names,upper_month_names)

      /***********************************/
       /* Determine user's primary choice: */
      /***********************************/
if ~(arg_supplied) then
  /* Pre-select "One Month", and ask user */
  /*  which choice he wants.              */
  /* Note: SelectFromList doesn't 'take'  */
  /* less than 4 as height (3rd arg).     */
  type = pdm_SelectFromList(,
       'Choose type',12,4,2,"_One Month"||cr||"Year")

      /*****************************/
       /* Choice is "One Month"   */
      /*****************************/
if type = "One Month" then
  do
    if ~(arg_supplied) then
      do
       default = substr(date('s'),5,2)
       if substr(date('s'),5,2) < 10 then
         default = substr(date('s'),6,1)
       default = default||'/'||subword(date(),3,1)
       month = pdm_GetForm("Enter month(mm/yyyy)",8,,
                           "MM/YYYY:"||default)
      end
    parse var month start_month'/'year
    args = "" /* if an error then want to interact with
                 user */
    if (start_month < 1 | start_month > 12) then
      do
       user_says = pdm_Inform(2,,
             "Month has to be 1...12","EXIT","RESTART")
       if user_says = 1 then signal restart
        else exit
      end
    RealYear = year
    year = FindAliasYear(year)  /* get a known year */
    /* get settings from the user-defined variables */
    height = month_height
    day_char_height = (72*height/6) *,
                 (.25+month_char_pos*.45)
     day_char_y =         (height/6) *,
                 (.3+(month_char_pos*.4))
   month_name_height = month_name_height*height*12
    width = month_width
    start_x = month_start_x
    start_y = month_start_y
    line_thickness = month_line_thickness
   call init_day_names(~short_day_name,monday_start,,
                       upper_day_names)
    print_year = 1 /* add year number to calendar */
    call DrawMonth
  end

      /*****************************/
       /* Choice is "Year"        */
      /*****************************/
 if type = "Year" then
  do
    if ~(arg_supplied) then
      year = pdm_GetForm("Enter year (yyyy)",5,,
                         "Year:"||subword(date(),3,1))
    args = "" /* if an error then want to interact with
                 user */
   call pdm_InitText(font,year_title_size*72)
    dummy_obj = pdm_DrawRectangle(0,0,0,0)
    call pdm_FlowText(year,3.2,0.75,6,0)  /* this is an
      undocumented call; I don't know what 5th argument
      does! (but it is required) As well, the documented
      TEXT() does not work! (arguments wrong order & ?)*/
   call pdm_SetFillPattern(,1,text_color,,,)
    call pdm_DeleteObj(dummy_obj)
    day_char_height = (72*year_month_height/6) *,
                      (.25+year_char_pos*.45)
     day_char_y =       (year_month_height/6) *,
                      (.3+(year_char_pos*.4))
    day_name_height = year_day_name_height
     RealYear = year
     year = FindAliasYear(year)  /* get a known year */
     height = year_month_height
    month_name_height = year_month_name_height*height*12
     width = year_month_width
     start_x = year_start_x
     start_y = year_start_y
     line_thickness = year_line_thickness
      start_month = 1   /* start on January */
      print_year = 0    /* don't add year number to
                           each month's calendar */
     /* center the text with respect to the boxes */
     one_more = ~year_3by4
    year_center =start_x+((3+one_more)*width+,
                          (2+one_more)*year_column_sep)/2
    call LRCenter(dummy_obj+1,year_center)
     call pdm_UnselectObj()
      call
init_day_names(~year_short_day_name,monday_start,,
                          upper_day_names)
     short_day_name = year_short_day_name
      do 4*year_3by4+3*~year_3by4     /* 3 or 4 */
        do 3*year_3by4+4*~year_3by4   /* 4 or 3 */
          call DrawMonth
          call pdm_UnselectObj()
          start_month = start_month +1
         start_x=start_x+year_month_width+year_column_sep
         end
        start_x = year_start_x
       start_y = start_y+year_month_height+year_row_sep
      end
   end
call pdm_UpdateScreen(1) /* only needed when called as */
                         /* a Genie                    */
/* print and/or quit */
select
  when print_after_drawing = 0 then
    Printed = 0
  when print_after_drawing = 1 then
    do
     call pdm_SetBatchMode(1) /* no requester */
     interpret print_string
      Printed = 1
     call pdm_SetBatchMode(0) /* requesters on */
    end
  when print_after_drawing = 2 then
    do
     user_says =pdm_Inform(2,"Print it now?","Yes","No")
      if user_says = 0 then
        do
         interpret print_string
          Printed = 1
        end
       else
        Printed = 0
    end
  otherwise
    Printed = 0
end
```

```
/* code taken out: printing not waited for
   AND delay() doesn't!
if Started_PDraw & Printed then
  do
    call pdm_Pause(20)  /* wait for a while (not long  */
                        /* enough to finish printing...*/
    call pdm_PDQuit()
  end
*/

exit /* Terminate program */

/*-----------------------------*/
/*    Subroutines and Functions:                        */
/*-----------------------------*/


/*-----------------------------*/
/* Draws a whole month. Can have from 4 to 6 partial or*/
/*  full weeks, depending on actual calendar.          */
/*-----------------------------*/
DrawMonth:
 first_dow = FirstDay(year,start_month,monday_start)
 first_day_no = date('c',,
                     10000*year+100*start_month+1,'s')
 call pdm_SetLineWeight(,line_thickness)
 call pdm_SetLineColor(,main_color)
  call pdm_SetFillPattern(,0,)              /* no fill */
 call pdm_SetLineJoin(,2)   /* Docs wrong:
                               2 args; 2 is bevel) */
 call pdm_InitText(font,month_name_height)
 dummy_obj = pdm_DrawRectangle(0,0,0,0)
 up_y = day_month_space*(height/6)*day_name_height
  if print_year then
   call pdm_FlowText(MonthName.start_month||' '||,
                     RealYear,start_x,start_y-up_y,10,0)
  else
  call pdm_FlowText(MonthName.start_month,start_x,,
                      start_y-up_y,10,0)
  if center_month_names then
   call LRCenter(dummy_obj+1,start_x+width/2)
 call pdm_DeleteObj(dummy_obj)
 if start_month=12 then
    do
    last_dow =FirstDay(year+1,1,monday_start)
    last_day_no = date('c',10000*(year+1)+101,'s')
    end
  else
    do
    last_dow =FirstDay(year,start_month+1,monday_start)
     last_day_no = date('c',,
                     10000*year+100*start_month+101,'s')
    end
 last_dow = last_dow-1;if last_dow == 0 then last_dow=7
month_length = last_day_no-first_day_no-1
no_weeks = (first_dow-1+month_length+7)%7
 del_y = height/6
 del_x = width/7
 left = start_x
 right = start_x+width
 /* Put on the day-names */
call pdm_InitText(font,72*day_name_height*del_y)
 do i = 1 to 7
  if center_day_names then
     do
      if short_day_name then
       x_corr=del_x/2-.375*day_name_height*del_y
         /*  .5 *.75 */
       else
       x_corr=del_x/2-1.125*day_name_height*del_y
         /* 1.5 *.75  */
      call pdm_FlowText(DayName.i,,
        start_x+((i-1)*del_x)+x_corr,,
         start_y-.05,,
         start_x+10,0) /* the +10 is arbitrary */
      end
     else
      call pdm_FlowText(DayName.i,,
       start_x+((i-1)*del_x)+del_x/32,,
        start_y-.05,,
        start_x+10,0) /* the +10 is arbitrary */
  end
  top = start_y; day_no = 1

 call pdm_InitText(font,day_char_height)
 if month_char_pos then /* centered */
  x_day_off = .5*del_x-.008*day_char_height
 else
   x_day_off = .1*del_x
 /* First row */
 call DrawDays(left,top,del_x,del_y,first_dow-1,,
             8-first_dow,day_no,,
            x_day_off,right_justify_numbers,day_char_y)
 /* Middle rows */
 day_no = day_no + 8-first_dow
 do i = 1 to no_weeks-2
   top = start_y+del_y*i
  call DrawDays(left,top,del_x,del_y,0,7,day_no,,
            x_day_off,right_justify_numbers,day_char_y)
   day_no = day_no + 7
 end
 /* Last row */
 top = top+del_y
 call DrawDays(left,top,del_x,del_y,0,last_dow,day_no,,
            x_day_off,right_justify_numbers,day_char_y)
 /* the next call affexts all the text*/
 call pdm_SetFillPattern(,1,text_color,,,)
 call pdm_SetLineWeight(,text_thickness)
return


/*-----------------------------*/
/* Draws one to seven days. Can be left or right just- */
/*  ified as required.                                 */
/*-----------------------------*/
DrawDays: procedure
arg s_x,s_y,d_x,h,d_skip,d_tot,start_day,x_day_off,,
   right_justify_numbers,day_char_y
    l_side =  s_x +  d_skip        * d_x
    r_side =  s_x + (d_skip+d_tot) * d_x
  objID = pdm_DrawRectangle(l_side,s_y,r_side,s_y+h)

   call pdm_SetFillPattern(objID,0,,,,)
    do j = 1 to 6-(7-d_tot)
      call pdm_InitPlot()
     objID = pdm_PlotLine(l_side+(d_x*j)" "s_y",",
                          l_side+(d_x*j)" "s_y+h)
    end
    call pdm_EndPlot()
    day = start_day
    do j = 1 to 7-(7-d_tot)
     if (day<10 & right_justify_numbers) then
       day_string = ' '||day
     else day_string = day
     call pdm_FlowText(,
    day_string,l_side+(d_x*(j-1))+x_day_off,,
     s_y+day_char_y,L_side+(d_x*j),10)
      day = day + 1
    end
   call pdm_EndPlot()
return


/*-----------------------------*/
/* Determine if the argument is a leap year:          */
/*-----------------------------*/
Leap:
arg yearin
```

```
  if (yearin//400 == 0) then return 1
  if (yearin//100 ~= 0 & yearin//4 == 0) then return 1
  else return 0


/*-------------------------------*/
/* Find the day of the week of the first day of the    */
/*  argument year:                                     */
/*-------------------------------*/
FirstDOY:
arg TargetYear
  if TargetYear < 1583 then
    do
     user_says = pdm_Inform(2,,
     "Years earlier than 1583 are"cr,
     "pre-Gregorian and hence not"cr,
     "drawn by this program.","EXIT","RESTART")
     call pdm_SetBatchMode(1)
     call pdm_DeleteObj() /* get rid of year title
                             if its there */
     if user_says = 1 then signal restart
      else
        exit
    end
  FirstDay = 7 /* 1978 started on a Sunday */
  if TargetYear < 1978 then
    do      /* count down to earlier years */
     do TestYear=1977 to TargetYear by -1
       if Leap(TestYear) then FirstDay = FirstDay-2
          else FirstDay = FirstDay-1
       if FirstDay = 0  then FirstDay = 7
       if FirstDay = -1 then FirstDay = 6
      end
    end
  else
    do      /* count up to later years */
     do TestYear=1979 to TargetYear
       if Leap(TestYear-1) then FirstDay = FirstDay+2
          else FirstDay = FirstDay+1
       if FirstDay = 9 then FirstDay = 2
       if FirstDay = 8 then FirstDay = 1
      end
    end
return FirstDay

/*-------------------------------*/
/* Return a year that the Amiga Rexx date call can deal*/
/* with which has the same calendar as the desired year*/
/*-------------------------------*/
FindAliasYear:
arg SearchYear
  StartDOW = FirstDOY(SearchYear)
  if Leap(SearchYear)=1 then
     do
       select
         when StartDOW = 1 then return 1996
         when StartDOW = 2 then return 1980
         when StartDOW = 3 then return 1992
         when StartDOW = 4 then return 2004
         when StartDOW = 5 then return 1988
         when StartDOW = 6 then return 2000
         when StartDOW = 7 then return 1984
         otherwise return 99
       end
     end
   else
     do
       select
         when StartDOW = 1 then return 1979
         when StartDOW = 2 then return 1985
         when StartDOW = 3 then return 1986
         when StartDOW = 4 then return 1981
         when StartDOW = 5 then return 1982
         when StartDOW = 6 then return 1983
         when StartDOW = 7 then return 1978
         otherwise return 98
       end
     end


/*-------------------------------*/
/* Calculate the first day-of the week for a month     */
/*  specified by the first two arguments: year, month. */
/* Can produce monday or sunday starts                 */
/*-------------------------------*/
FirstDay: procedure
arg yr,mon,monday_start
  if datatype(mon) ~= 'NUM' then
     do
      l_s = translate(left(mon,3)) /* ucase left-most 3*/
       select
         when l_s = 'JAN' then mon = 1
         when l_s = 'FEB' then mon = 2
         when l_s = 'MAR' then mon = 3
         when l_s = 'APR' then mon = 4
         when l_s = 'MAY' then mon = 5
         when l_s = 'JUN' then mon = 6
         when l_s = 'JUL' then mon = 7
         when l_s = 'AUG' then mon = 8
         when l_s = 'SEP' then mon = 9
         when l_s = 'OCT' then mon = 10
         when l_s = 'NOV' then mon = 11
         when l_s = 'DEC' then mon = 12
       end
     end
 l_s = date('w',(10000*yr)+(100*mon)+1,'s')
  dow = DayOfWeek(l_s,monday_start)  /* Sunday Start */
return dow

/*-------------------------------*/
/* Convert the text returned by the date fn to an index*/
/*  number.                                            */
/*-------------------------------*/
DayOfWeek: procedure
arg str,flag
/* figure out day number:
 *  1...7 for Mon...Sun if flag = 1
 *  1...7 for Sun...Sat if flag = 0
 */
 str = translate(left(str,2))
 select
   when str = "MO" then dow = 1
   when str = "TU" then dow = 2
   when str = "WE" then dow = 3
   when str = "TH" then dow = 4
   when str = "FR" then dow = 5
   when str = "SA" then dow = 6
   when str = "SU" then dow = 7
 end
 if ~flag then
  do
    dow = dow +1
      if dow = 8 then dow=1
  end
return dow


/*-------------------------------*/
/* Initialize the variables DayName.1 ... DayName.7    */
/*-------------------------------*/
init_day_names:
parse arg long,monday_start,UC
if monday_start then
  if long then do
    DayName.1="Mon"
    DayName.2="Tue"
    DayName.3="Wed"
    DayName.4="Thu"
```

```
    DayName.5="Fri"
    DayName.6="Sat"
    DayName.7="Sun"
  end
  else do
    DayName.1="M"
    DayName.2="T"
    DayName.3="W"
    DayName.4="T"
    DayName.5="F"
    DayName.6="S"
    DayName.7="S"
  end
else
  if long then do
    DayName.1="Sun"
    DayName.2="Mon"
    DayName.3="Tue"
    DayName.4="Wed"
    DayName.5="Thu"
    DayName.6="Fri"
    DayName.7="Sat"
  end
  else do
    DayName.1="S"
    DayName.2="M"
    DayName.3="T"
    DayName.4="W"
    DayName.5="T"
    DayName.6="F"
    DayName.7="S"
  end
if long & UC then
  do i = 1 to 12
   DayName.i = translate(DayName.i)
  end
return

/*-------------------------------*/
/* Initialize the variables MonthName.1...MonthName.12 */
/*-------------------------------*/
init_month_names:
arg long,UC
do
 MonthName.1 ="January"
 MonthName.2 ="February"
 MonthName.3 ="March"
 MonthName.4 ="April"
 MonthName.5 ="May"
 MonthName.6 ="June"
 MonthName.7 ="July"
 MonthName.8 ="August"
 MonthName.9 ="September"
 MonthName.10="October"
 MonthName.11="November"
 MonthName.12="December"
 end
if ~long then
  do i = 1 to 12
   MonthName.i =left(MonthName.i,3)
  end
if UC then
  do i = 1 to 12
   MonthName.i = translate(MonthName.i)
  end
return

/*-------------------------------*/
/* Move an object horizontally so that its center is   */
/*  at the coordinate provided in the second argument. */
/*-------------------------------*/
LRCenter: procedure
arg object,hor_ctr
dummy_obj = pdm_DrawRectangle(0,0,0,0)
if pdm_IsText(object) then
  do
   call pdm_UnselectObj()
   text = pdm_SelectObj(object,dummy_obj-1)
   call pdm_GroupObj()
  end
call pdm_DeleteObj(dummy_obj)
objpsn = pdm_GetCenter()
parse var objpsn objx objy
objsze = pdm_GetObjSize()
parse var objsze wid hgt
call pdm_SetObjPosn(,hor_ctr-wid/2,objy-hgt/2)
return

/*-------------------------------*/
/* Similar to Gold-disk provided routine: modified to  */
/*  work with CLI if PDraw isn't running.              */
/*-------------------------------*/
exit_msg: procedure expose units PD_open
do
 parse arg message
 if PD_open then
   do
    if message ~= '' then call pdm_Inform(1,message,)
    call pdm_SetUnits(units)
    call pdm_AutoUpdate(1)
   end
 else
  say message
 exit
end

/*-------------------------------*/
/* Three error traps. All call exit_msg, above.        */
/*-------------------------------*/
error:
 exit_msg("Function failed due to error: ",
        errortext(rc)" line: "sigl)
syntax:
 exit_msg("Function failed due to sytax error: ",
        errortext(rc)" line: "sigl)
break_c:
 exit_msg("User aborted ")
```

✔

The complete set of listings can be found on the AC's TECH disk.

Please write to:
Keith Brown
c/o AC's TECH
P.O. Box 2140
Fall River, MA 02722

1993 AC's TECH Development Contest WINNERS!

The premise of the 1993 Development Contest was to create an application or hardware modification to allow disabled users better access to the Amiga. The entries were tested and the winners chosen for their creativity and the integrity of their creations. Two winners were chosen. They will each receive a copy of the AMOS Development System and a free one-year subscription to *AC's TECH*.

And the winners are:

R.P. Haviland of Daytona Beach, FL

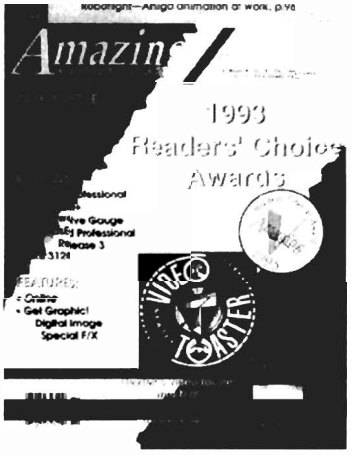
Mr. Haviland sent a collection of utilities designed to aide the visually impaired user. The first utility, KEYREAD, pronounces the name of each key depressed and on hitting the space bar, reads the word which was just typed. The second utility, LINEREAD, reads all the words in a line until it reaches a return. The line then may be repeated or a new line read. His third utility, READFILE, is a simplified version of LINEREAD. It will read an entire file but without the options available in LINEREAD. Good work Mr. Haviland.

Gary Smith of Brisbane, Australia

Mr. Smith submitted a unique program which replaces the keyboard with the mouse. This allows the user to type in a word processing or other program by using the mouse instead of the keyboard. Excellent for users who are unable to navigate a keyboard.Very good Mr. Smith.

Congratulations to the winners and thank you all for your entries. It is good to see that the Amiga community is concerned with the interests of disabled Amiga users.

Three ways to make your life easier:



Amazing COMPUTING™ / AMIGA®

Amazing Computing For The Commodore Amiga is dedicated to Amiga users who want to do more with their Amigas. From Amiga beginners to advanced Amiga hardware hackers, AC consistently offers articles, reviews, hints, and insights into the expanding capabilities of the Amiga. *Amazing Computing* is always in touch with the latest new products and new achievements for the Commodore Amiga. Whether it is an interest in Video production, programming, business, productivity, or just great games, AC presents the finest the Amiga has to offer. For exciting Amiga information in a clear and informative style, there is no better value than *Amazing Computing* .

AC's TECH / AMIGA®

AC's TECH For The Commodore Amiga is the first disk-based technical magazine for the Amiga, and it remains the best. Each issue explores the Amiga in an in-depth manner unavailable anywhere else. From hardware articles to programming techniques, *AC's TECH* is a fundamental resource for every Amiga user who wants to understand the Amiga and improve its performance. *AC's TECH* offers its readers an expanding reference of Amiga technical knowledge. If you are constantly challenged by the possibilities of the world's most adaptable computer, read the publication that delivers the best in technical insight, *AC's TECH For The Commodore Amiga.*

AC's GUIDE / AMIGA®

AC's GUIDE is a complete collection of products and services available for your Amiga. No Amiga owner should be without *AC's GUIDE.* More valuable than the telephone book, *AC's GUIDE* has complete listings of products, services, vendor information, user's groups and public domain programs. Don't go another day without *AC's GUIDE*!

Live better with Amazing Computing

1-800-345-3360

SUBSCRIBE!

YES! *The "Amazing" AC publications give me* ***3 GREAT*** *reasons to save!*

Please begin the subscription(s) indicated below immediately!

Name_____

Address _____

City_____ State _____ ZIP_____

Charge my Visa MC #_____

Expiration Date _____ Signature _____

Please circle to indicate this is a **New Subscription** or a **Renewal**

MasterCard® VISA DISCOVER

Call now and use your Visa, Master Card, or Discover or fill out and send in this order form!

1 year of **AC**	12 big issues of Amazing Computing! Save over 43% off the cover price!	**US $27.00** ☐ **Canada/Mexico $34.00** ☐ **Foreign Surface $44.00** ☐
1-year SuperSub	AC + AC's GUIDE—14 issues total! Save more than 45% off the cover prices!	**US $37.00** ☐ **Canada/Mexico $54.00** ☐ **Foreign Surface $64.00** ☐
1 year of AC's TECH	4 big issues of the FIRST Amiga technical magazine with Disk!	**US $43.95** ☐ **Canada/Mexico $47.95** ☐ **Foreign Surface $51.95** ☐

Please call for all other Canada/Mexico/foreign surface & Air Mail rates.

Check or money order payments must be in US funds drawn on a US bank; subject to applicable sales tax.

ORDER!

NAME_____

ADDRESS_____

CITY_____STATE_____ZIP_____

CHARGE MY: ___VISA ___M/C #_____

EXPIRATION DATE_____ SIGNATURE_____

MasterCard® VISA DISCOVER

Amazing Computing Back Issues: $5.00 each US, $6.00 each Canada and Mexico, $7.00 each Foreign Surface. Please list issue(s)_____

Amazing Computing Back Issue Volumes:
Volume 1-$15.00* Volume 2, 3, 4, 5, 6, 7, or 8—$20.00* each or any 12 issues for $20.00*
* All prices now include shipping & handling. * Foreign surface: $25. Air mail rates available.

AC's TECH/AMIGA **Single issues just $14.95! V1.1 (PREMIERE), V1.2, V1.3, V1.4, V2.1, V 2.2, V2.3, V2.4, V3.1, V3.2, V3.3, V3.4**

Volume One, Two, or Three (complete) or any four issues— $40.00!

Freely Distributable Software – Subscriber Special (yes, even the new ones!)

1 to 9 disks	**$6.00 each**
10 to 49 disks	**$5.00 each**
50 to 99 disks	**$4.00 each**
100 or more disks	**$3.00 each**

$7.00 each for non subscribers **(three disk minimum on all foreign orders)**

Amazing on Disk:

AC#1 ...Source & Listings V3.8 & V3.9
AC#2 ...Source & Listings V4.3 & V4.4
AC#3 ...Source & Listings V4.5 & V4.6
AC#4 ...Source & Listings V4.7 & V4.8
AC#5 ...Source & Listings V4.9
AC#6 ...Source & Listings V4.10 & V4.11
AC#7 ...Source & Listings V4.12 & V5.1
AC#8 ...Source & Listings V5.2 & 5.3
AC#9 ...Source & Listings V5.4 & V5.5
AC#10 ...Source & Listings V5.6 & 5.7
AC#11 ...Source & Listings V5.8, 5.9 & 5.10
AC#12 ...Source & Listings V5.11, 5.12 & 6.1
AC#13 ...Source & Listings V6.2 & 6.3
AC#14 ...Source & Listings V6.4, & 6.5
AC#15 ...Source & Listings V6.6, 6.7, 6.8, & 6.9

Back Issues: $_____

AC's TECH: $_____

PDS Disks: $_____

Total: $_____

(subject to applicable sales tax)

Please list your Freely Redistributable Software selections below:

AC Disks _____
(numbers 1 through 15)

AMICUS _____
(numbers 1 through 26)

Fred Fish Disks _____
(numbers 1 through 910)

Complete Today, or telephone 1-800-345-3360 now!

You may FAX your order to 1-508-675-6002

Please allow 4 to 6 weeks for delivery of subscriptions in US.

(Domestic and Foreign air mail rates available on request)

Check or money order payments must be in US funds drawn on a US bank; subject to applicable sales tax.

VideoStage Pro™

TITLING, SOUND SYNCHING, DAZZLING TRANSITIONS

VideoStage Pro offers an innovative, intuitive approach to titling videos, creating transitions between video or graphic segments and sound synching. Individual characters or whole lines of text or objects can be flown on to the screen. Automatic detection for "hot Colors" in both NTSC and PAL prevents bleeding colors. Gradient backdrops, gradient text and transparency options adds to the polished appearance of output. The Story Board builds shows by clicking on event icons. A time line graphic represents the duration of events such as sound and transitions. VideoStage Pro offers over 60 built- in transitions available for use with a click of the mouse. Play Control indexes can be selected with the mouse to create play loops and "Hot Spots" allow for fully interactive on-screen presentations such as kiosks, training,etc. Titles can utilize all Amiga fonts including color fonts and Compugraphic fonts. VideoStage Pro can be remotely controlled through ARexx, modems or networks. Asynchronous control of genlocks and sound allow for quick, easy creation of videos with sound.

VideoStage Pro is compatible with AmigaDOS 3.0 and the new AGA chipset

VideoStage Pro List Price $179.95

Upgrades are available for Video Titler and AniMagic users from Oxxi

SBASE™ PROFESSIONAL

Data isn't just text and numbers anymore. The Amiga computer opened up the world of graphics and sound. SBase (formerly known as Superbase) helps you keep track of your pictures, sound files and anims so you can readily retrieve them or harness them for creative applications.

With use of graphic files - you can create a database of inventory not just by part number but by a graphic image as well

Sound samples can be added to impart special directions or simply to help you tie in pictures and sounds used on a project.

SBase's full relational capabilities and intuitive interface makes it one of the most powerful database's on any platform. Capacities of database files and indexes are limited only by disk storage and your creativity.

Version 1.3 of SBase adds compatibility with AmigaDOS 3.0 and the new AGA chip set, use of Anim files, EPS clip art and compugraphic re-scalable fonts. The new Re-index feature gets you out of jams fast.

S/BASE™ PERSONAL

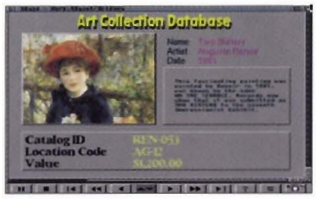
SBase Personal 4 includes a built-in text editor, mail merge, label printing, and form designer.

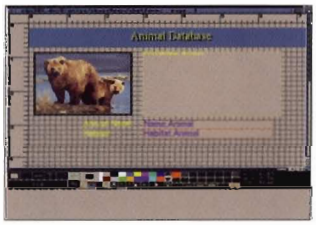
SBase Professional 4 includes all of the features of the Personal version plus adds support for ARexx and the Database Management Language (DML) for creation of custom applications.

SBase 4 developer's extension is a one time license that provides the ability for applications developed with DML to run by themselves without requiring the user to use the full blown SBase Professional 4.

SBase Personal 4	-	List	**$149.95**
Sbase Professional 4	-	List	**$299.95**
SBase 4 Developers Extension			**$399.95**

Upgrades and Updates from earlier versions of Superbase Personal and Professional are available from Oxxi - call or FAX for information

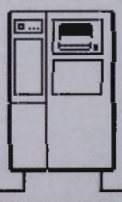




Networking with Novell Netware AMIGA CLIENT SOFTWARE™

With Amiga Client Software (ACS), Amiga computers and Video Toaster workstations enjoy all network capabilities available to PCs and Macintoshes on Novell Networks. Amiga workstations retain their full multi-tasking and graphical environment in addition to receiving the full range of Novell Netware functions. All computers on the network can then share files and peripherals such as large hard disks and printers.

ACS allows for data integrity including record locking of shared files and flagged files. Includes utilities for backup and restoration of server from Amiga workstations and broadcasting and display of messages to either the whole network , work groups, or individual users.

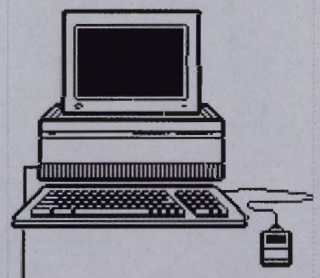


Support for IPX Protocol allows applications that support IPX to communicate directly with other Amigas on the network.

ACS is priced according to the number of Amiga users on the network starting with a single user for $199 and up to 5 users for $499. For larger configurations please call.

Ethernet Cards

Oxxi is shipping Ethernet Cards that accept both co-axial and 10 Base-T. List Price $359



Dazzling On Screen, Slide or Hard Copy Presentations PRESENTATION MASTER™

Presentation Master integrates object-orientated painting, business graphics, text handling and 3D titling all using 24 bit color and Postscript output.

Object Orientated Painting

Presentation Master's structured paint system converts all paint objects whether text, outline fonts and imported clip art into Bezier curves for detailed object editing and manipulating. Objects can be grouped and ungrouped. Automatic object tweening and morphing allows the user to alter text or clip art images.Support for Illustrator EPS clip art provides access to millions of high-quality, ready-to-use clip art images. Over 100 clip art images included on the program.

24 Bit Color and Business Charts

Create images for use with DC-TV, Digital Micronics Resolver Board and Video Toaster and other 24 -bit hardware and software. Use Master template styles provided with Presentation Master or create your own templates for professional quality output. Includes two disks of backdrops as well.

Standalone Shows

Any shows can be converted into self-running presentations complete with on screen "hot spots" for audience interactive presentations.

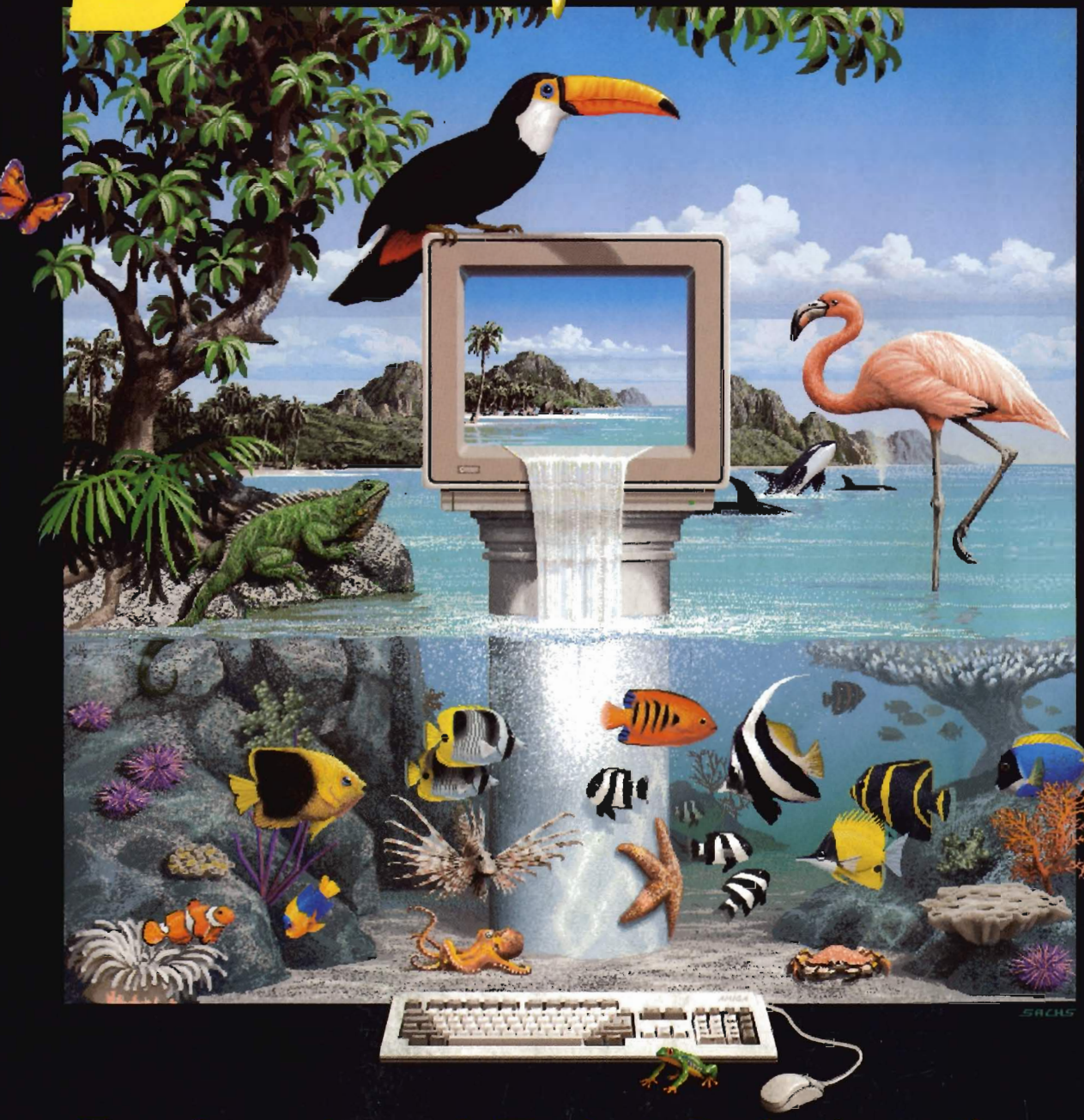
List Price: $299.95

Updates from earlier versions available from Oxxi

Oxxi inc.
PO Box 90309, Long Beach, CA 90809
(310) 427-1227 • FAX: (310) 427-0971

IT'S SHIPPING!

BRILLIANCE



Professional Paint & Animation

DIGITAL CREATIONS